Real-Time Occupancy Grid Mapping using LSD-SLAM

by

Graham Hull

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electrical Engineering in the Faculty of Engineering at Stellenbosch University

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Dr. C.E. van Daalen

December 2017

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2017

Abstract

Real-Time Occupancy Grid Mapping using LSD-SLAM

G. Hull

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MScEng (Mech)

March 2017

This thesis investigates the use of semi-dense depth data from monocular vision using large scale direct SLAM (LSD-SLAM) to create accurate occupancy grid maps for autonomous navigation, in real-time. Having an accurate map is crucial for an autonomous system to avoid collisions and remain safe within its environment. Sensors used to gather information on the environment are typically associated with some degree of uncertainty, and this must be considered when building a map. An autonomously navigating system also needs to have clear definition of free and occupied space within its environment.

Literature shows that LSD-SLAM has great potential as a highly accurate and real-time SLAM algorithm; however, the resulting map is in the form of a semi-dense point-cloud which is not immediately useful to an autonomously navigating system. The point-cloud map must therefore be processed further.

Occupancy grid maps (OGMs) offer an ideal solution for map representation that is useful for autonomous navigation. The environment is divided into evenly spaced grid cells, each representing a probability of occupancy. OGMs also allow maps to be efficiently updated with the incorporation of uncertainty from sensor measurements.

Inverse sensor models (ISMs) can be used to characterise the uncertainty of a particular sensor and to calculate the prediction of occupancy given a sensor measurement and its uncertainty. A literature review shows that two popular ISMs (one by Thrun and one by Andert) can be used in conjunction with the depth estimates of LSD-SLAM to create an OGM. Literature also shows that each of these ISMs contains a parameter that is associated with very little information on how their values should be chosen, and we therefore included this in our investigation.

We design a mapping system using the aforementioned ISMs, which runs in parallel with LSD-SLAM. Initial tests show that the performance of the open-source version of LSD-SLAM did not agree with the author's claims. The results also revealed a significant lack of sufficient datasets for our main evaluations on map accuracy.

Our mapping system was tested on 3 main criteria: memory usage, performance and accuracy. All evaluations were performed on both ISMs, on various datasets, over a range resolutions and parameter changes for each ISM. Results showed that Thrun's ISM out-performed Andert's ISM on all criteria, and that our system could indeed produce accurate maps that could be useful for autonomous navigation. The results also showed that the "default" choice for the parameters of each ISM is not necessarily always sufficient. Additionally, we conclude that LSD-SLAM does not perform well in terms of 30 Hz real-time requirements, while our mapping system can.

Uittreksel

Intydse Besetting-Rooster-Kaarte deur die gebruik van LSD-SLAM

("Real-Time Occupancy Grid Mapping using LSD-SLAM")

G. Hull

Departement Elektriese en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MScIng (Meg)

Maart 2017

Hierdie tesis ondersoek die gebruik van semi-digte diepte data van monovisie deur gebruik te maak van groot skaal direkte SLAM (LSD-SLAM) om akkurate besetting-rooster-kaarte vir outonome navigasie intyds op te stel. Om 'n akkurate kaart te hê is krities vir 'n outonoome stelsel om botsings te vermy en om veilig te bly in die omgewing. Sensors wat gebruik word om inligting oor die omgewing in te samel, word tipies geassosieer met 'n mate van onsekerheid, en dit moet in ag geneem word wanneer 'n kaart gebou word. 'n Outonome navigasiestelsel moet ook duidelike definisie van vrye en besette ruimte binne sy omgewing hê.

Literatuur wys dat LSD-SLAM groot potensiaal het as 'n hoogs akkurate en intydse SLAM algoritme, maar die resulterende kaart is egter verteenwoordig as 'n semi-digte punt-wolk, wat nie onmiddellik bruikbaar is vir outonome navigasie stelsels nie. Die punt-wolkkaart moet dus verder verwerk word.

Besetting-rooster-kaarte (OGMs) bied 'n ideale oplossing vir kaartvoorstelling wat nuttig is vir outonome navigasie. Die omgewing word verdeel in eweredig verspreide roosterselle, wat elkeen 'n waarskynlikheid van besetting verteenwoordig. OGMs laat ook toe om kaarte doeltreffend op te dateer met die insluiting van onsekerheid van sensormetings.

Inverse sensor modelle (ISM's) kan gebruik word om die onsekerheid van 'n spesifieke sensor te karakteriseer en die voorspelling van besetting te bereken wat 'n sensormeting en sy onsekerheid gegee het. 'n Literatuuroorsig toon dat twee gewilde ISM's (een deur Thrun en een deur Andert) saam met die diepte ramings van LSD-SLAM gebruik kan word om 'n OGM te skep. Literatuur toon ook dat elkeen van hierdie ISM's 'n parameter bevat wat geassosieer word met baie min inligting oor hoe hul waardes gekies moet word, en ons het dit dus by ons ondersoek ingesluit.

Ons ontwerp 'n kartering-stelsel met behulp van die voorafgemelde ISMs, wat in parallel met LSD-SLAM uitvoer. Aanvanklike toetse toon dat die uitvoering van die oopbron weergawe van LSD-SLAM nie met die skrywer se eise ooreenstem nie. Die resultate het ook 'n aansienlike gebrek aan voldoende datastelle vir ons hoofevaluasies op kaart akkuraatheid geopenbaar.

Ons kartering-stelsel is getoets op 3 hoofkriteria: geheueverbruik, spoed en akkuraatheid. Alle evaluasies is uitgevoer op beide ISM's, op verskeie datastelle, oor 'n reeks resolusies en parameter veranderinge vir elke ISM. Resultate het getoon dat Thrun se ISM Andert se ISM op alle kriteria uitpresteer het, en dat ons stelsel inderdaad akkurate kaarte kan produseer wat nuttig kan wees vir outonome navigasie. Die resultate het ook getoon dat die "standaard" keuse vir die parameters van elke ISM nie noodwendig altyd voldoende is nie. Daarbenewens kom ons tot die gevolgtrekking dat LSD-SLAM nie goed presteer in terme van 30 Hz intyds vereistes, terwyl ons kartering-stelsel kan.

Acknowledgements

I would like to express gratitude and appreciation to:

- My supervisor for his guidance and insight
- My colleagues for their tolerance and support
- My family and friends for their love and patience

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Efficient autonomous robotic navigation and mapping have been a goal for many decades. In recent years this has become a rapidly growing field with many significant advances being made. Whether it is for autonomous quadrotor drones, exploring harsh environments (on Earth or other planets), or autonomous cars, autonomous navigation and mapping has become essential. With the increase in efficiency of mass production and computing power, robotic resources have become more readily and cheaply available for development. For example, the cost of cameras and computing power has shown steadily improving trends over the past few decades [16; 17; 18].

Autonomous robots need to be able to safely navigate unknown environments, without the assistance of a human. Generally this involves using sensors to extract information from the robot's environment to create a map that can be used to make navigational decisions based on collision avoidance.

Autonomously navigating robots typically employ simultaneous localisation and mapping (SLAM) as the basis for navigation. This involves the estimation of the robot's pose based on the map, while creating the map based on the robot's pose, at the same time. The classic analogy of the chicken-and-egg conundrum is mostly associated with this problem. This challenging problem has been of high interest to researchers in this field and can be regarded as solved on a theoretical level [19], although more work needs to be done to address the challenges of real-life SLAM [20].

To measure the environment, sensors must be employed, and many different types of sensors are suited for different purposes. Acoustic sensors, such as sonar, are more likely to be used in underwater applications where there is poor visibility. 2D laser sensors are fast and precise, making them a very attractive option. While laser sensors typically only operate in a 2D plane, they can be mounted on a controlled rotation gimbal or servo to incorporate the third dimension. This does, however, add to the complexity of combining scans which is subject to numerous errors. Vision sensors such as stereo or monocular cameras can be used to estimate the 3D structure of the environment by using disparity or motion parallax respectively.

An advantage that single cameras have over stereo systems is that one camera is always cheaper and lighter than two. Another advantage, is the simplicity of the installation and calibration for monocular vision versus stereo vision. Having a SLAM system work with a single camera allows for a larger range of robotic platforms to gain autonomous navigation features, especially smaller and lighter platforms such as drones where weight is a major concern.

Recently, the Technical University of Munich (TUM) unveiled its large scale direct SLAM (LSD-SLAM) algorithm that is capable of performing real-time SLAM with monocular vision to create semi-dense depth maps with highly accurate pose estimations [4]. The authors claims that it can even run as odometry on a modern smart-phone, which indicates the algorithm's potential efficiency and ability to use multi-core processors. This is a big step forward in that it allows the use of a single low cost camera to achieve good results for localisation and mapping of the environment in the form of a semi-dense 3D point-cloud. Another major advantage is that it has been made open-source [25] which allows the field to grow more significantly, especially for development in the low cost sector.

One caveat to this is that two versions of LSD-SLAM are available. One that is made freely available to the public and a proprietary commercial version that is for sale by TUM. It can be argued that a third version also exists, that is used as an in-house version for research purposes by TUM [26; 25]. This distinction makes it clear that results obtained during testing may differ from what is presented in the original LSD-SLAM paper [4]. This is discussed further at the end of Chapter 3. In this thesis, we will only consider the open-source version, so that continuation and extension of this project can be done without further commercial arrangement.

LSD-SLAM is a standalone system that only provides a map in the form of semi-dense point-cloud. This is not immediately useful to an autonomously navigating robot in terms of path or motion planning since navigation requires explicit definition of free and occupied space. Therefore the point cloud data requires further processing.

When it comes to path or motion planning of a robot, it is essential to be aware of the environment, or to have an accurate map of the environment. Knowing the location and size of obstacles are key to avoiding them, as well as knowing where else the robot can move if a specific path is blocked. An occupancy grid map is extremely useful for this purpose since it allows a metric map with occupancy information to be stored and used.

1.2 Problem Statement and Strategy

The goal of this thesis is to investigate the use of LSD-SLAM and monocular vision to create a 3D occupancy grid map from semi-dense depth data of a static environment, in real time. The problem will be approached in a general sense, in that a specific application will not be considered during development of the project, but a better understanding of applications will be discussed after testing. Therefore it is assumed that the robot or camera may be controlled by either a human or a path planning algorithm.

The main input sensor to the overall system will be a monocular camera. This is ideal for robotic systems since they are low cost, light weight and simple to implement. The LSD-SLAM algorithm will use the camera images to get semi-dense probabilistic depth data by performing many small variable baseline stereo comparisons between key frame images and consecutive image frames. The depth data is output as an array of inverse depth values with associated inverse depth variances. The LSD-SLAM algorithm also provides an accurate pose estimation of the camera that is further refined once a loop closure is detected and completed. The depth data from the LSD-SLAM algorithm will be applied to an inverse sensor model and will be used along with the camera pose to construct the occupancy grid maps in real time. In order to free up memory resources, the resolution of the resulting 3D occupancy map will be adapted to the requirements of the depth data. This is discussed in Chapter 4.

LSD-SLAM was developed to run in the well known Robot Operating System (ROS) [44]. ROS was developed by Willow Garage to provide many tools and libraries to enable software developers to make robotics applications. It is very widely used not only in simulation, but on embedded systems as well. Therefore ROS will be used to develop this system since there are also already well established libraries, communities and documentation for the incorporation of occupancy grid maps. The system will also be developed to be as open-source as possible (in accordance with the original LSD-SLAM GNU General Public licensing V3 [26] and BSD License for ROS [44]) and as 'plug and play' as possible, so that further testing could be done with minimal effort.

The input data will be monocular datasets provided by TUM. The datasets will be selected from a large library on the basis of getting a large variance in scenes and settings. This will allow us to evaluate and discuss some of the strengths and weaknesses of LSD-SLAM and the resulting occupancy grid map, as well as the combined system.

The tests will be done in accordance with two criteria; *qualitative* and *quantitative* on the basis of evaluation for path planning purposes. The quality of the maps will be compared to unprocessed maps without ground truth, showing visuals of their differences with and without the use of an inverse sensor model. Unprocessed maps are considered to be point clouds that are not filtered through a sensor model and dumped into an occupancy grid map.

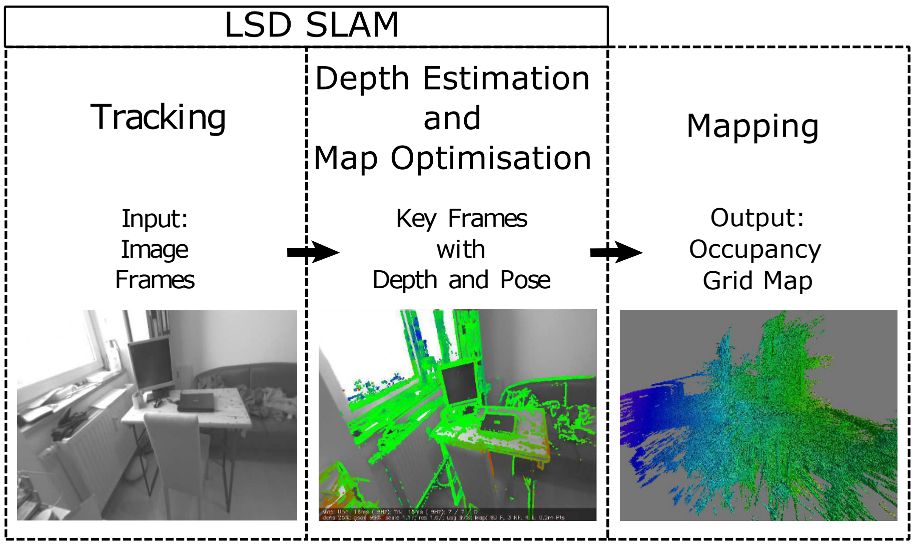


Figure 1.1: Overview of project system on the Robotics Operating System (ROS) with 3 main nodes. LSD-SLAM Node with the camera input (left), the Depth Estimation Node (centre) and the Occupancy Grid Mapping Node (right).

This is necessary since there are datasets that work specifically well for LSD-SLAM, that do not have ground truth. The more important quantitative testing entails a comparison of the resulting map to a ground truth to evaluate the error between the two. In terms of path planning it is essential that the maps provide a good indication of free space and immediate obstacle detection. The actual path planning implementation within a map is out of the scope of this thesis. As mentioned above and will be discussed further later, the inherent problem of the lack of scale in monocular vision alone does not allow for the resulting map to be to scale either. This can be corrected with sensor fusion of an Inertial Measurement Unit (IMU), however, this is also out of the scope of this thesis.

We also aim to investigate the usefulness of the resulting map for obstacle avoidance and path planning. If the map provides enough information based on semi-dense input for autonomous navigation, then a system that uses a single camera to create the map in real time would prove to be an important step forward in he field of autonomous navigation and robotics. There are a number of problems faced when evaluating the resulting maps which will be discussed further in Chapter 6.

1.3 System Overview

As mentioned above, the system will be developed using ROS. The system is divided into 3 nodes that send messages along a chain, as shown in Figure 1.1.

As input, monocular vision data will be fed into the LSD-SLAM Tracking

Node, where the LSD-SLAM algorithm will perform small variable baseline stereo comparisons to estimate the depth of pixels with sufficient gradient information. Many images are used to update the depth estimate of one pixel, and therefore the baseline stereo comparisons continuously change. The pose will be estimated by a Gauss-Newton minimisation algorithm when using direct image alignment in the tracking process. The Tracking Node and the Depth Estimation Node exchange data to perform their respective calculations, with the goal of producing an optimised map of the environment with accurate pose estimates. The output of the Depth Estimation Node will be an array (one array for every key frame) of inverse depth estimates for the relevant pixels, along with the depth variances of those depth estimates, as well as the pose estimates of that key frame in SIM(3) format. This output data is used by the Mapping Node to create the occupancy grid map. The depth data is applied to an inverse sensor model and can be further optimised for memory conservation by adapting the resolution and calculating the maximum likelihood of the resulting map.

1.4 Outline of Thesis

This thesis starts by providing an overview of previous literature in Chapter 2 that covers the relevant fields necessary for the understanding of this thesis. This offers motivation for the design and approach of our proposed system, as well as any assumptions that are made. In Chapter 3 we go into more detail on how LSD-SLAM works and what limitations need to be overcome. Chapter 4 discusses occupancy grids along with some derivations, followed by some derivations and explanations of inverse sensor models in Chapter 5. We then discuss the resulting system with some tests to compare two inverse sensor models in Chapter 6 and finally end with our conclusion and recommendations for future work in Chapter 7.

Chapter 2

Literature Review

In this chapter we discuss some relevant research and important concepts from literature to provide a background on which our research is based. In Section 2.1 we discuss the basic concept of SLAM, and particularly in the context of monocular vision in Section 2.2. We give a brief introduction to monocular vision and some common methods of monocular SLAM with reference to *tracking and mapping* methods in Section 2.2.3, followed by a brief overview of LSD-SLAM and what makes it different to common monocular SLAM methods in Section 2.2.5. Section 2.3 discusses maps and map representation, specifically occupancy grid maps and the use of local, large and global maps in Section 3.5. And finally in Section 2.4 we briefly discuss path planning and obstacle avoidance. These discussions shape how the goals of this project are approached and why.

2.1 SLAM

2.1.1 Basic Concept

For a robot to navigate its environment, it is necessary for it to know its current location or pose in either 2D or 3D space. In most cases this involves using the environment and identifiable or trackable features within the environment, to estimate the robot's pose. However, the location of these features also needs to be known, and are usually measured and estimated using the robot's relative pose. Simultaneous localisation and mapping (SLAM) involves placing a robot in an unknown location, and then creating a consistent map of an unknown environment while estimating its pose, at the same time [11].

Sensors on the robot are used to detect and measure features or obstacles in the environment and sonar, laser and vision sensors are commonly used for this purpose. The choice of sensors depend on the application and the requirements of the robot [45]. Sonar often provides noisy and simplified data which may be enough to avoid large obstacles, however the materials of the

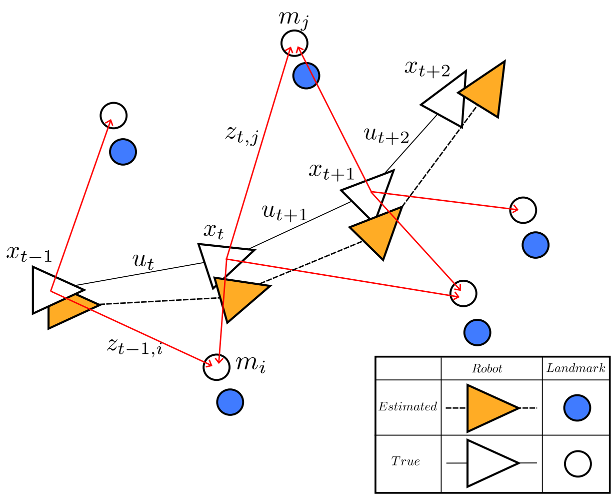


Figure 2.1: The essential SLAM problem based on the image by [1]. The robot is placed in an unknown environment at an unknown location, and attempts to estimate the location of landmarks, as well as its own location within the environment. It will never know the true location of the landmarks it observes or its own pose within the environment.

environment play a large role on the quality of measurements [46]. Laser range sensors may provide much more accuracy and resolution, but the amount of data becomes costly (in time and resource) to process [47]. Vision sensors such as stereo or monocular cameras can also provide dense measurements, depending on the method used or the application requirements [5]. Regardless of the sensor used, sensor noise is always inherent in the system (as well as noise from other factors such as the environment or imperfect control mechanisms) meaning the estimates of the robot pose and map will also contain noise. For this reason approaching SLAM estimates using probabilistic methods proves to be extremely useful [11].

Consider the situation in Figure 2.1, where a mobile robot is placed in an unknown environment at an unknown location, and as it moves through the environment it uses its sensors to observe and measure landmarks relative to its own location, where:

x_t is the state vector of robot's pose (location and orientation) at time t

u_t is the control vector that was applied at the previous time step $(t-1)$ to move the robot to position x_t

m_i is the location of a single landmark i of which the true location is assumed to be independent of time t. A vector of landmarks (m) is used to described the map of the environment.

z_{it} is the vector of measurements made from the robot to the location of landmark i at time t

From a probabilistic point of view, SLAM can be separated into two problems, online SLAM and full SLAM [27; 11]. Online SLAM is concerned with estimating the robot's pose and a map of the environment for the current time step, which is in accordance with the Markov assumption. Generally this involves making an estimate of the present state based on the directly preceding state of the robot. Once the present state is estimated, it is considered independent of past or future states. Full SLAM attempts to estimate a posterior distribution over the entire pose trajectory of the robot ($x_{1:t}$), along with the map m. Contrary to online SLAM, Full SLAM is generally not incrementally solved. Full SLAM is formulated as

$$p(x_{1:t}, m|z_{t:1}, u_{1:t}). \tag{2.1.1}$$

As mentioned above, online SLAM estimates the posterior over the robot's current pose x_t, along with the map m. This gives the following probabilistic solution,

$$p(x_t, m|z_{1:t}, u_{1:t}). \tag{2.1.2}$$

Under the Markov assumption, it is assumed that past and future information are independent if the current state is known. In other words it is assumed that all information needed for the current state estimation is included in the estimation from the previous time step. For online SLAM this assumption can be applied to estimate the current state, given the previous state estimate and measurement, and control state. Therefore Equation 2.1.2 becomes

$$p(x_t, m|x_{t-1}, z_t, u_t). \tag{2.1.3}$$

In terms of online SLAM, the extended Kalman filter (EKF) is arguably the most widely used state estimation algorithm. It is an extension of the Kalman Filter allowing it to be used for non-linear systems. The Kalman Filter uses many observations and measurements over time to estimate an unknown state, while incorporating uncertainty and statistical noise into the observations and measurements. Generally the Kalman Filter uses Bayesian estimations to solve the SLAM problem, and uses Gaussian distributions to represent posterior probabilistic states of the robot [27]. EKF SLAM creates a large vector of sensor and landmark states to represent the map. These states are modelled by a multivariate Gaussian distribution. The map is continuously updated on every time step by prediction (when the robot is instructed to move) and correction (when previously seen landmarks are measured again by the sensors) processes [48; 11; 49].

An alternative to EKF SLAM, which uses Gaussian distributions to model uncertainty, is a Monte Carlo method that can be used to model distributions with a set of sample points (particle filter). However, unlike Gaussian distributions, particle filters are subject to the problem of high dimensionality. To combat this, an assumption is made that if the robot's pose is known, then all landmark features are independent of each other, i.e. feature locations are only dependent if there is uncertainty in the robot's pose. This assumption allows for the Rao-Blackwellized particle filter to be used to create the Fast-SLAM algorithm, where the posterior over the robot's states are represented by a particle, and a Gaussian distribution method (such as an EKF) is used to represent all other variables such as landmarks [11].

Initially the full SLAM algorithm, FastSLAM1.0, was developed to combat data association failures in EKF during prolonged sensor updates of large numbers of landmarks, but proved to be inefficient when generating particles. FastSLAM1.0 also suffered with poor estimation accuracy of the robot's pose since it only used control inputs to predict its location. This lead to the development of FastSLAM2.0, which incorporated a motion model as well as sensor measurements to improve pose estimation accuracy. FastSLAM2.0 outperforms FastSLAM1.0, but when a large number of landmarks are present, it suffers in runtime efficiency [50; 1].

This thesis is focused on real-time applications and therefore the discussion would naturally be narrowed down to online SLAM. However since the LSD-SLAM method is based on monocular vision and therefore subject to drift (this will be discussed in further detail in Section 2.2.4), a global optimisation algorithm is used to correct for this drift which relies on past measurements and states. This global correction is described as a Full SLAM implementation. LSD-SLAM works on the premise of tracking (for localisation) and mapping simultaneously using a monocular sensor. The following section will discuss monocular SLAM and some related methods, followed by a brief overview of LSD-SLAM.

2.2 Monocular SLAM

Landmarks in the environment can be measured or identified in many different ways. They can have specific identifiers or features like corners, straight lines, intensity gradient, or simple distance measurements. The choice of features or landmarks to measure are dependent on the algorithm used, resources available and the application of the robot. Vision based sensors can provide significantly more data per frame than alternative sensors such as sonar or laser, although the challenge then lies in processing the large amount of data in an efficient manner. The main decision that needs to be made during the development of a vision based SLAM system, is what type of features to track between incoming frames [51]. A set of unordered images of the same scene or the motion of a camera can be used to estimate the structure of the environment, a process commonly known as *structure from motion* (SFM) [52; 53]. The development of monocular vision SLAM systems started with stereo vision SLAM systems. This binocular vision allows for the depth of the scene to

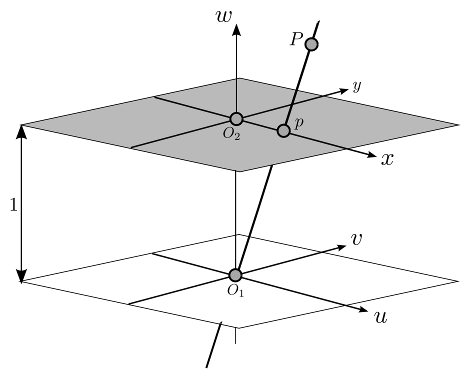


Figure 2.2: Basic visual representation of homogeneous coordinates, where point P is projected onto the common plane $w = 1$ at point p.

be estimated; however, they may require very precise hardware setup and calibration. Monocular vision solves this problem, but not without the cost of the lack of scale. With monocular vision alone, there is no way to determine absolute metric scale of the environment, which will be discussed further in Section 2.2.4. In this thesis we will focus specifically on monocular vision.

2.2.1 Homogeneous Coordinates

Before we describe the camera model in the following section, a brief overview on the homogeneous coordinate system should be discussed. Homogeneous coordinates are used to represent Euclidean geometry in the form of projective geometry, which allows for infinitely far points to be represented by finite coordinates, and also allows for the use of a single matrix to represent affine transformations as well as projective transformations. For the representation of a geometric object or point P, it is considered homogeneous if P and λP (where λ is a scalar) represent the same object for $\lambda \neq 0$.

Figure 2.2 shows a 3D point P along a common vector that goes through the origin O_1. Point p is point P projected onto the common plane $w = 1$, which is further described by,

$$P = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{2.2.1}$$

Any 2D vector can be represented in homogeneous coordinates by projecting it onto the common plane $w = 1$ as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{2.2.2}$$

and this can easily be extended to 3D.

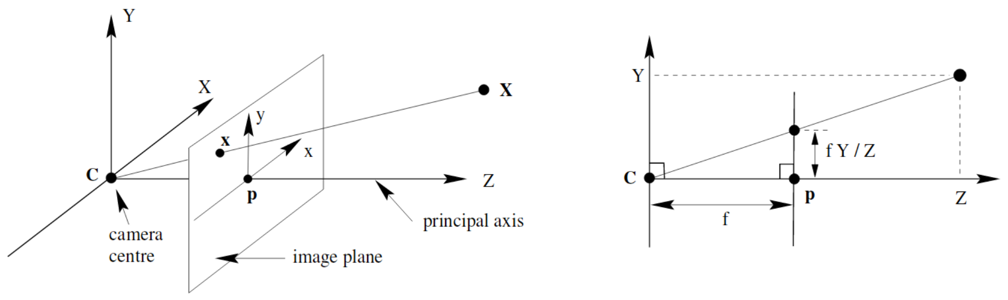


Figure 2.3: Basic geometry of pinhole camera model [2]. Note the principle axis from camera center C to point P on the image plane.

2.2.2 Camera Model

Because we have chosen to make vision sensors our primary source of measurement, it is important to know how to use them to gather useful information about the environment. Cameras provide many image frames per second, most commonly at approximately 30 Hz at a resolution of about 640 x 480 pixels. Each frame contains information to describe the scene, such as RGB colour, grayscale intensities or sometimes depth as in the case of a Microsoft Kinect [54].

Each image frame is a 2D representation of the 3D environment. The goal is to reverse this projection from the 2D plane back into 3D world space. This can be achieved once a model describing the conversion from 3D coordinates to 2D coordinates is known, such as the well known pinhole camera model shown in Figure 2.3, where:

f is the focal length containing components f_x and f_y to represent the x and y axis respectively. This is used as the "scaling" factors from 3D to 2D.

C is the camera center, which is not necessarily in the center of the 2D image plane. It contains components c_x and c_y to represent the camera center offset in the x and y axis of the image plane respectively. C does have a 3D pose (translation and rotation) in real world coordinates.

Together these parameters form a calibration matrix K, where

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2.3}$$

Consider the 3D point X in Figure 2.3 represented by a vector of homogeneous coordinates, where

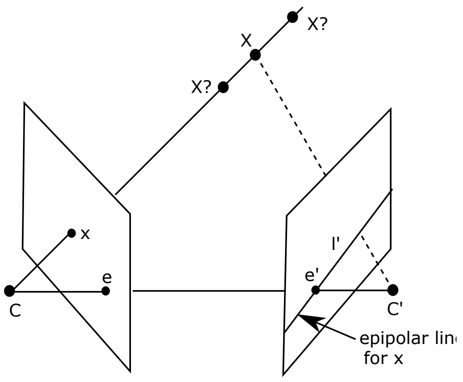


Figure 2.4: Basic epipolar geometry of a stereo camera model [2]. The real world point X projects a 2D point on each of the camera's image planes. A baseline between camera centres C and C' is projected through their respective image planes in this case by e and e' for the left and right camera planes respectively. A line is drawn between e' and l' which creates an epipolar plane on which point X must lie.

$$X = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{2.2.4}$$

The conversion from the 3D space to 2D space is then described as,

$$[K|0] \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} f_x x + c_x z \\ f_y y + c_y z \\ Z \end{pmatrix}. \tag{2.2.5}$$

With the basic camera model described, it is important to note that monocular vision SLAM algorithms employ stereo vision techniques to gain depth information on the environment. Figure 2.4 shows that using the camera model in a typical stereo set up with consecutive frames of the same point X, it is possible to calculate the real 3D location. Assuming the two frames come from two separate camera locations, the camera centres can be linked by a baseline. The baseline between the two camera centres create an intersection line between the two image frames to create a plane known as the epipolar plane that also intersects both frames. From the perspective of the cameras, this epipolar plane intersection is just a line. By creating an epipolar line between two images of the same point, it narrows down the search area for that point in the other image. In the case of Figure 2.4 with respect to camera C', the point x' will lie somewhere on the line between point x and camera C. These epipolar constraints make depth calculations more efficient and accurate.

Once a point correspondence has been made between two images, and assuming the baseline distance between the two camera centres are known, a simple disparity calculation can be made to determine the distance of the point relative to the subject frame. Disparity (shown in Equation 2.2.6) describes the difference in distance between the projected points x and x' on the image planes for the cameras of the 3D point X in the real world, where B is the baseline distance between camera centres and z is the distance to 3D point X.

$$\text{disparity} = x - x' = \frac{Bf}{z} \tag{2.2.6}$$

2.2.3 Monocular SLAM Methods

MonoSLAM, presented by Davison [51], produces a probabilistic feature-based map while keeping track of the pose estimates, as well as uncertainty of both the map and pose estimates, with a single camera. It was the first monocular SLAM implementation, with the added advantage of running in real time (30 Hz), albeit only for small environments. The EKF based algorithm allowed a sparse map to be updated with new information, while also updating or deleting old information. This provided a good platform for real time monocular vision research to be launched from.

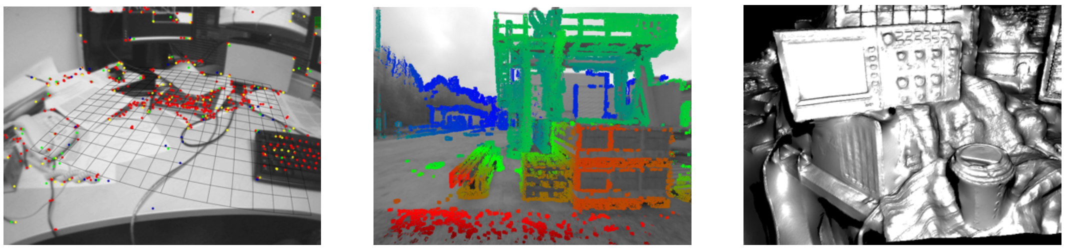
PTAM (parallel tracking and mapping) provided a new monocular SLAM method specifically designed for hand-held augmented reality purposes, by dividing the system into two processes that run in parallel threads on a multi-core processor [3]. The tracking process handled the robust pose estimation of the camera with the assumption that the provided map is perfect. The mapping process, which runs at the same time, attempts to optimise thousands of points in the map by making stereo comparisons between new image frames aligned with a chosen key frame. The mapping process also makes the assumption that the pose estimate is perfect. The key frames are chosen based on criteria such as being a certain distance away from the previous key frame, being the 20th frame after a key frame or based on tracking quality. This allowed the algorithm to produce a map with more detail since map updates are only done on each new key frame; however, the resulting map is mainly a sparse point cloud.

DTAM (dense tracking and mapping) became the next step in research over the previous sparse methods of PTAM and MonoSLAM [5]. DTAM used PTAM as a base input, but with a dense depth estimation hypothesis over all pixels for every key frame. Not only does DTAM create dense depth maps, but also exhibits better tracking quality than PTAM since it is more robust to motion blur.

LSD-SLAM is another method for monocular SLAM, and is the main method on which we focus during the course of this thesis. LSD-SLAM works on a similar principle to PTAM and DTAM in that it contains a tracking and mapping component that run in parallel; however, it falls in between the two methods in terms of information density as a semi-dense method. Further discussion on LSD-SLAM is provided in its own section in Section 2.2.5.

For the above mentioned tracking and mapping algorithms (PTAM, DTAM and LSD-SLAM), the assumption of correctness is of course in reality not true since the map and pose estimations are subject to some errors. Some sources of error are sensor noise, translational and rotational drift, as well as scale drift.

It is worth noting here that the brief overview of the monocular SLAM methods mentioned above, are based on a distinct characteristic. The quantity of data used in the algorithm is classified as being either dense, semi-dense or sparse as shown in Figure 2.5. Dense vision based SLAM algorithms use all information in the image to reconstruct the scene, where as semi-dense algorithms use smaller portions of information that are defined by some criteria. The criteria may be characterised by distinct features in the scene such as lines, corners or thresholds of pixel intensities, although overall, enough information is used to summarise the reconstructed scene. Sparse algorithms generally only provide a point-cloud-like map based on key features or points, and is mostly used for camera localisation. The distinction between *sparse* and *semi-dense* is vague and without a concrete definition, however for the purposes of this thesis we will only consider *dense* or *semi-dense* implementations.



(a) Sparse (b) Semi-Dense (c) Dense

Figure 2.5: Quantities of data used in 3 monocular SLAM methods showing, sparse point data from the PTAM algorithm (a) [3], semi-dense pixel data from LSD-SLAM (b) [4] and dense pixel data from DTAM (c) [5].

2.2.4 Loop Closure and Scale Drift

As mentioned above, SLAM algorithms assume correctness of calculated poses and maps. This causes a challenge of translational and rotational drift in the pose and map. If a robot (using only vision-based SLAM) starts at a location and records its path through the environment, the resulting map and pose will degrade over time. Once the robot has returned to the starting position again, the resulting map will in all likelihood not accurately reflect this. One method of correcting for drift in pose and mapping estimates, is to detect and

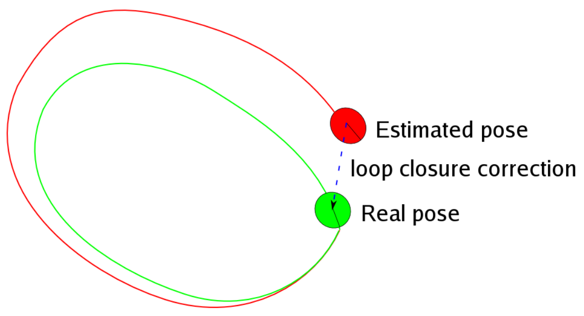


Figure 2.6: Basic loop detection and closure, where red shows the incorrect pose estimation and green shows the corrected pose estimation [6].

perform a loop closure. If a scene has been visited before and it is detected by an algorithm, it can use this information to correct for the drift using an optimisation process [6]. The entire recorded path travelled by the robot can be adjusted until its perceived starting location is realigned with its actual starting location, as shown in Figure 2.6.

Loop closure can also correct for scale drift. An inherent problem with monocular vision is the inability to detect absolute scale, which is made worse by movement. This is because even though small stereo comparisons are made between incoming frames, the stereo baseline is not perfectly known and changes depending on the motion of the camera [55]. The perceived scale will change over time as the camera, or robot, moves through an environment [51; 4].

2.2.5 LSD-SLAM

Large scale direct SLAM (LSD-SLAM), follows a similar approach to PTAM and DTAM in that it contains a tracking and mapping component that runs in parallel on two separate threads. However unlike PTAM, LSD-SLAM uses a featureless approach, making it a *direct method*.

For a feature-based (indirect) approach, two main steps are required: feature extraction and feature matching [3; 48]. Features in an image can be edges, corners or more complex descriptors, and can be detected using techniques such as SIFT (scale invariant feature transform) or FAST (features from accelerated segment test). The aim with feature detection is to identify salient or distinct parts of an image, which are then described with a descriptor to allow the feature to be recognised from various viewpoints of the same image scene. Since feature-based algorithms remove any data that is not considered a feature, it makes these algorithms very fast in implementation as there is much less data to process. The disadvantage of feature-based methods is that the

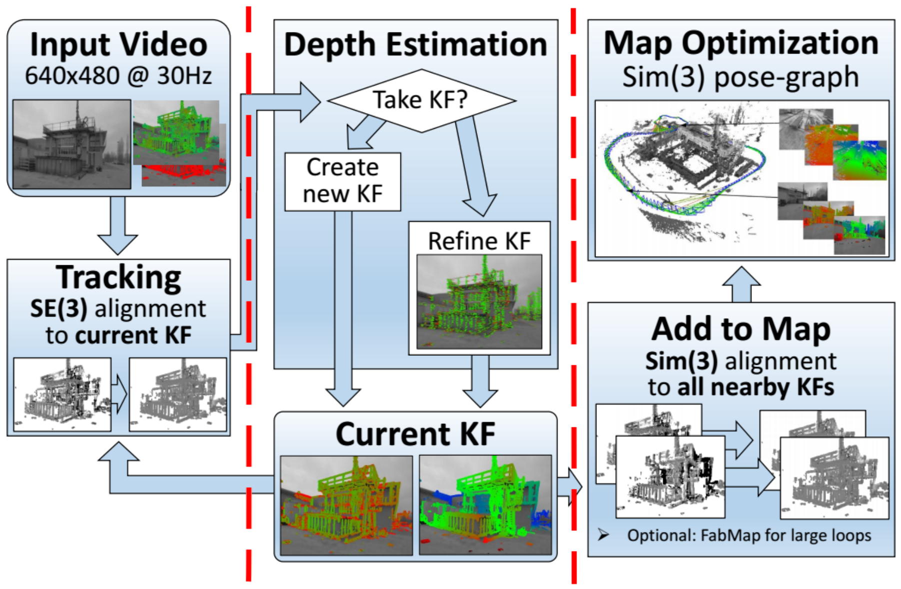


Figure 2.7: An overview of the process flow of LSD-SLAM. The overview can be divided into three columns to represent the three main processes of the algorithm: tracking, depth estimation and map optimisation respectively [7]

algorithm requires that the scene being observed have features that conform to the *feature type* being used by the algorithm [4]. It is possible to create a large database of feature types to fit most scenes, however this becomes tedious when switching between completely different environment types [56; 57]. Feature-based algorithms are more suited to corners or key points in an environment scene and can therefore miss out on important or useful information in man-made environments where there are more straight lines and curved edges. Man-made environments also often lack texture (in terms of what is visually apparent within an environment), which feature-based methods often rely on.

Direct methods do not rely on features, but rather make use of pixel intensities within the image. It can use the entire image of a key frame, and compare it to another incoming frame to create a more meaningful representation of the environment. Because it works directly on the image pixels and does not need to store processed features, it can create semi-dense 3D maps in real time [4; 58].

The disadvantage of direct methods is that they do not handle outliers efficiently, and are slower than feature-based approaches [7].

As mentioned, LSD-SLAM contains the familiar Tracking and Mapping component. However the mapping component can be split into two separate processes; one for depth estimation and one for map optimisation. An overview of the process flow is shown in Figure 2.7.

As seen in Figure 2.7, LSD-SLAM can be divided into 3 main parts; tracking, depth estimation and map optimisation. The tracking component continuously estimates the camera's 3D pose using direct image alignment between the current key frame and new incoming frames. The image alignment is performed by minimising the photometric error, or difference in pixel intensities, between frames.

The depth estimation component is responsible for either refining or replacing the depth calculations for the current key frame, based on some criteria such as a minimum real world coordinate distance away from the pose of the previous key frame. During the refinement of a key frame's depth, depth estimation is performed by computing many efficient small baseline stereo comparisons on a per-pixel level for each consecutive incoming frame. The pixels used for depth estimation are chosen based on their intensity gradients, i.e., the difference in brightness between a pixel and its neighbour in a particular direction. LSD-SLAM first converts any RGB image to gray scale, making these intensity gradients trivial to access. Once a new key frame is chosen, the previous key frame is regarded as complete and the depth estimation for that key frame is no longer refined. The depth estimation for a key frame results in a point cloud, or an array of depth values for every pixel, relative to the key frame at its calculated pose.

The map optimisation component incorporates a completed key frame into a global map. The resulting map consists of a set of key frames with estimated poses, each with their own depth estimates. A loop detection and closure process also takes place to correct for any drift that occurs during the mapping process. A similarity transform using scale-aware direct SIM(3) image alignment, is estimated between key frames, along with an appearance based large scale loop closure algorithm. The similarity transform is performed on every key frame in comparison to nearby key frames (usually up to 10), including the key frame prior to the current. However larger loop closures are detected by an appearance based algorithm called OpenFABMap (open-source fast appearance based mapping), which is an open-source toolbox for this purpose [59]. The resulting map is a pose graph of key frames, which is essentially a point cloud with interconnected key frames at their respective pose estimates.

2.3 Map Representation

In general, there are two main types of maps: topological maps and metric maps. Street maps or bus route maps are examples of topological maps. These kinds of maps store places, or nodes, with links or pathways between them in the form of graph representations as shown in Figure 2.8(a). They may prove useful in situations where path planning is the main concern since paths between nodes are already mapped out, and the symbolic representations make for efficient computations. However it is often difficult to find optimal paths between nodes as the geometry of the environment is not accurately described in the map [30].

Metric maps represent the environment more accurately than topological maps. Metric maps contain accurate proportions or measurements to describe the environment. Grid based maps, are a form of metric map, that generally contain evenly spaced grid cells as shown in Figure 2.8(b). The grid cells may represent a portion of the environment that contains an obstacle or free space, or if it is still unknown. Grid based maps are simple to construct and store a lot more detail about the environment than topological maps. They also allow for more accurate solutions for optimal paths between locations, although the added complexity of large amounts of information takes up significantly more memory and makes path planning computations less efficient.

Metric maps, and specifically occupancy grid maps, are the most common type of maps used in autonomous navigation and robotics [28]. Occupancy grid maps allow metric maps to be built on the principle of probabilistic theory and will be discussed in further detail in Chapter 4.

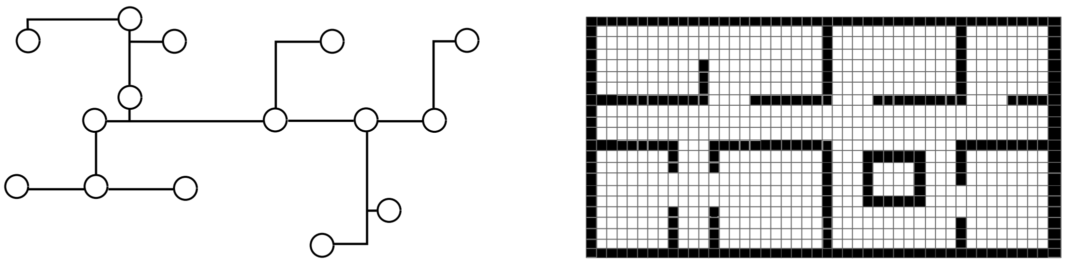

(a) Topological Map (b) Metric Map

Figure 2.8: Map representations of office space showing a topological map (a) with paths between nodes and a metric map (b) with more geometric detail of the environment [8].

2.4 Path Planning and Obstacle Avoidance

Autonomously navigating robots need to be able to get from one location to a goal location. To do this the robot needs to move along a path while trying to avoid hitting objects that it should not be colliding with. To do this efficiently, a path planning mechanism needs to be put in place. Topologically, the problem of path planning is related to finding the optimal path between two locations, or nodes, in a graph, and these algorithms are measured by their computational complexity. The optimal path could be considered as being the shortest distance, path of lowest energy consumption or the fastest path including obstacle avoidance [60].

There are a large range of classes of path planning algorithms in existence, each with their own set of requirements, strengths and weaknesses. At the stage of path planning it is generally assumed that the environment has already been measured and separated into classes of occupied or free space [61; 62]. Sample-based planning algorithms, such as rapidly exploring random trees (RRT) or probabilistic road maps (PRM), require that information on the environment exists as a set of nodes or cells depicting obstacles and free space. Broadly speaking these methods create a set of random paths to a goal location within an environment, and compare the resulting paths against the environment to determine if the path would involve a collision with an obstacle, or successfully avoids occupied space.

Node base planning optimal algorithms, such as Dijkstra's algorithm, A^* search and D^*, require that the environment be decomposed into a node graph before analysing possible paths [63]. Node based algorithms are search algorithms and, assuming the graph is finite, can always find an optimal path. These methods are widely used and are often capable of handling dynamic threats of collision.

Bio-inspired algorithms like genetic algorithms and neural networks are designed to allow for path planning without human intervention, and are modelled after humans and animals based on how they navigate a cluttered environment [64]. Simplistically, these algorithms start by creating a random set of feasible solutions, which are then evaluated against specific constraints such as the environment, the robot's state, the goal and more. The best solutions are then selected as *parents* of the next generation of solutions, with some mutation. This basic principle is performed iteratively until the best solutions are created.

No path planning algorithm can be considered the "best" and therefore opens a new possibility of combining algorithm strengths into a class of multi-fusion based algorithms. The general aim is to find a computationally efficient path planning algorithm that consistently finds a global optimal solution. Regardless of the method used to find an optimal path, a very important requirement is to know the geometry and metrics of the environment and obstacles to avoid [65]. The more accurate the map, the better it is for path planning and obstacle avoidance. In this thesis we are not dealing with path planning or obstacle avoidance algorithms, but aim to create a metric map suitable for these algorithms, primarily on the criteria of accuracy and performance.

Chapter 3

LSD-SLAM

3.1 Overview of LSD-SLAM

Large scale direct SLAM, or LSD-SLAM, is an algorithm that uses monocular vision to create a map of an unknown environment and to determine the pose of the camera within the created map. Many vision-based SLAM algorithms rely on features or landmarks in a scene to track camera movements and trajectories, which is where LSD-SLAM primarily differs by using the pixels on the camera images directly. Similarly to PTAM and DTAM, LSD-SLAM chooses key frames and estimates the depth over a subset of pixels by making many stereo comparisons with consecutive image frames. It is also designed to work in large scale environments.

In this chapter we will discuss some details on how LSD-SLAM works, based primarily on work by [4], with supporting discussion based on work by [58], [66], [67] and [26]. We will start with some preliminary information required to give the reader some background on key concepts before going into details on the core components of the algorithm. As mentioned in Chapter 2.2.5 and as shown in Figure 3.1, LSD-SLAM can be divided into three main components and will be discussed in order, as tracking in Section 3.3.1, mapping in Section 3.3.2 and map optimisation in 3.3.3. At the end of this chapter we provide some commentary, caveats and challenges that may be encountered using this algorithm.

3.2 Preliminary Information

3.2.1 Intensity Gradients

LSD-SLAM uses intensity gradients to identify pixels within an image frame, or more precisely, the difference in intensity gradients between neighbouring pixels, as well as the direction of the difference in intensity gradient. Pixels observed to have a sufficient intensity gradient are used for tracking compar-

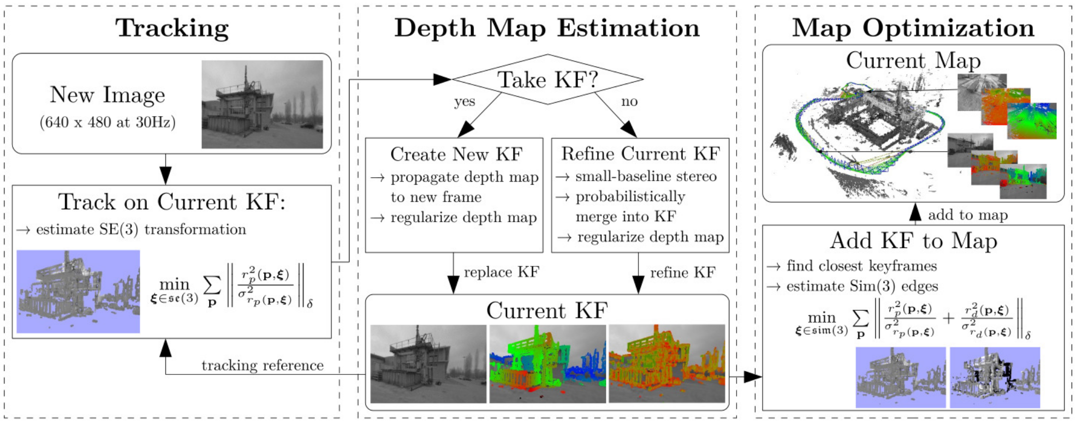

Tracking
New Image
(640 x 480 at 30Hz)
Track on Current KF:
→ estimate SE(3) transformation
$\min_{\xi \in \mathfrak{se}(3)} \sum_{\mathbf{p}} \left\| \frac{r_p^2(\mathbf{p},\xi)}{\sigma^2_{r_p(\mathbf{p},\xi)}} \right\|_\delta$
tracking reference
Depth Map Estimation
Take KF?
yes
no
Create New KF
→ propagate depth map to new frame
→ regularize depth map
Refine Current KF
→ small-baseline stereo
→ probabilistically merge into KF
→ regularize depth map
replace KF
refine KF
Current KF
Map Optimization
Current Map
add to map
Add KF to Map
→ find closest keyframes
→ estimate Sim(3) edges
$\min_{\xi \in \mathfrak{sim}(3)} \sum_{\mathbf{p}} \left\| \frac{r_p^2(\mathbf{p},\xi)}{\sigma^2_{r_p(\mathbf{p},\xi)}} + \frac{r_d^2(\mathbf{p},\xi)}{\sigma^2_{r_d(\mathbf{p},\xi)}} \right\|_\delta$

Figure 3.1: A visual representation of the three main components in the LSD-SLAM algorithm; Tracking, Depth Map Estimation and Map Optimisation [4].

isons between image frames. As mentioned in Chapter 2.2.5, the input images may be RGB colour images but are converted to monochrome which makes access and calculation of intensity gradients more efficient.

3.2.2 Key Frames

A *key frame* is described as being an image frame of significance in the LSD-SLAM algorithm. Key frames are normal image frames that have been promoted to a key frame based on the criterion of either being the very first image frame in the sequence, or being a frame that is more than a specified minimum distance away from a previous key frame. A key frame K_i is used to represent a set of frames within the specified minimum distance and has parameters described by Equation 3.2.1, where I_i is the *ith* key frame image, D_i is the inverse depth map of the image I_i and V_i is the array of inverse depth variances of inverse depth map D_i, as follows,

$$K_i = (I_i, D_i, V_i), \tag{3.2.1}$$

The depth map D_i is a 2D array of values that correspond to the pixels which contain significant intensity gradients. Figure 3.2 shows a visual representation of the pixels with sufficient intensity gradients used for the depth map of that key frame, which also shows the semi-dense nature of the algorithm. The inverse depth estimates are modelled and stored as a Gaussian distribution, as $\mathcal{N}(d, \sigma^2)$, with a mean inverse depth d and an inverse depth variance of σ^2.

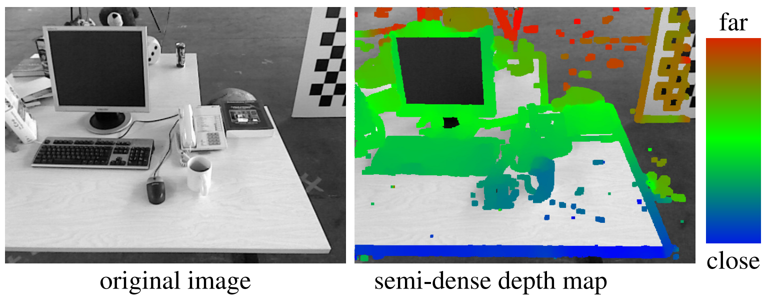


Figure 3.2: A visual representation of a key frame with pixels that have sufficient intensity gradients (coloured to encode distance) to be used for its depth map.

3.2.3 Pose Representation

LSD-SLAM uses two main representations for camera pose ξ, depending on the stage of the calculations within the algorithm. In the early image tracking stage of a key frame image, a 3D rigid body $\xi \in \mathrm{SE}(3)$ transformation representation is used. Later when map and pose optimisation occurs, a 3D similarity transform $\xi \in \mathrm{SIM}(3)$ is used. Both SE(3) and SIM(3) representations of pose are mainly used for just that: representation. However, LSD-SLAM makes extensive use of optimisation methods, and the pose representations do not work well for these types of methods since the resulting pose may not always be a valid transformation. Therefore the pose transformations are converted into Lie group/algebra form for optimisation calculations. In the context of this thesis and LSD-SLAM, the pose of a key frame is always relative to the pose of the previous key frame. The initial key frame to the dataset is always initialised as the origin of the world coordinate system.

3.2.3.1 SE(3) - 3D Rigid Body Transformation

SE(3) or Special Euclidean Group can be regarded as the "standard" method for pose representation, encoding six degrees of freedom as rotation R and translation t into a transformation T, where $R \in \mathrm{SO}(3)$ is the 3D rotation matrix and $t = [x, y, z]^T$ is the 3D translation vector as

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}. \tag{3.2.2}$$

3.2.3.2 SIM(3) - 3D Similarity Transformation

SIM(3) is similar to SE(3) but with scale as an added degree of freedom, where $s \in \mathbb{R}$ is the scale, as shown in Equation 3.2.3 as

$$T = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix}. \tag{3.2.3}$$

3.3 Details of LSD-SLAM

With the above preliminary information, we can proceed with a more detailed discussion of LSD-SLAM. In this section we will discuss each of the 3 main components of LSD-SLAM as tracking, depth estimation and mapping.

3.3.1 Tracking

The first stage of the LSD-SLAM algorithm is to take in an image from a camera. The input image may be an RGB image, although it will get converted to monochrome before processing. The very first image is made a key frame and is given a random inverse depth map with each relevant pixel containing a large mean and variance, where only pixels with sufficient intensity gradients are given an inverse depth value. Consecutive image frames are aligned to the key frame by directly minimizing the photometric error $E_p(\xi)$ between pixels to estimate the camera pose of the new image relative to the key frame image using the Gauss-Newton algorithm. This can be described by the following equation:

$$E(\xi) = \sum_i (I_{kf}(p_i) - I(\omega(p_i, D_{kf}(p_i), \xi)))^2 := \|r(\xi)\|_2^2, \tag{3.3.1}$$

where,

ξ is the 3D camera pose of the new frame relative to the key frame in SE(3),

p_i is the ith relevant pixel location of the image frame for which there is a depth,

$I(p)$ is the relevant pixel intensity of the key frame image I_{kf} or current image frame I,

$D_{kf}(p)$ is the inverse depth value of pixel p in the key frame image,

$\omega(p_i, D_{kf}(p_i), \xi)$ is the projection of pixel p with its associated inverse depth value to a new image frame, given the transform ξ.

In Equation 3.3.1 the accumulated residual $r(\xi)$ uses a warp function ω to project the relevant pixels, using the inverse depth map of the key frame D_{kf}, to the transformed perspective ξ. If each projected pixel corresponds exactly to the correct pixels in the key frame, then the pose estimation is exact and the difference in pixel intensity between these points would become zero. However an exact match will hardly be the case considering factors like image noise, and therefore an optimisation needs to occur to minimise this error in pose estimation. A weighted Gauss-Newton optimisation is used to find the optimal pose ξ^* of the camera to minimise the photometric error as,

$$\xi^* = \arg\min_{\xi} E_p(\xi). \tag{3.3.2}$$

As the residual for each pixel point p tends toward zero, the photometric error $E_p(\xi)$ tends toward zero, and therefore the estimated pose ξ becomes closer to being correct. Since depth is used in this calculation, and the depth is just an estimate, the reliability of this estimate needs to be taken into account as well. Therefore depth uncertainty is incorporated into the estimates by normalising the photometric error using the inverse depth variance V_i as shown by Equation 3.3.3:

$$E_p(\xi_{ij}) = \sum \left\| \frac{r_p^2(p, \xi_{ij})}{\sigma_{r_p}^2(p, \xi_{ij})} \right\|_\delta, \tag{3.3.3}$$

with

$$r_p(p, \xi_{ij}) := I_i(p) - I_j(\omega(p, D_i(p), \xi_{ij})) \tag{3.3.4}$$

and

$$\sigma_{r_p}^2(p, \xi_{ij}) := 2\sigma_I^2 + \left(\frac{\partial r_p(p, \xi_{ij})}{\partial D_i(p)}\right)^2 V_i(p), \tag{3.3.5}$$

where

σ_I^2 is the Gaussian distributed image intensity noise and $\|\cdot\|_\delta$ is the Huber norm.

The depth values are only estimates and therefore the quality of the estimate needs to be considered to gain a more reliable final depth estimate. The residual variance ($\sigma_{r_p}^2(p, \xi_{ij})$) is calculated using *covariance propagation* (described in detail by Engel et al. [4]) and the inverse depth variance V_i.

The Huber norm is applied to the normalised residual (Equation 3.3.6) to handle outliers in an attempt to make it more robust to occlusions and pixel noise.

$$\|r^2\|_\delta := \begin{cases} \frac{r^2}{2\delta} & \text{if } |r| \leq \delta \\ |r| - \frac{\delta}{2} & \text{otherwise.} \end{cases} \tag{3.3.6}$$

The pose estimation of an image frame is done over multiple resolutions from a coarse-to-fine approach to handle tracking of large motions of the camera. Starting at a high pyramid level (coarse resolution), the resulting pose estimation becomes the initial estimate for the next pyramid level (finer resolution) and so on. The final pose estimation of the finest pyramid level becomes the initial pose estimate for the following image frame. Once a frame is tracked and the pose has been estimated, a decision is made to either use it to refine the depth of the current key frame image, or to promote it to become the new key frame. This decision is based on the distance between the current key frame and the current image frame, and the number of pixels that were

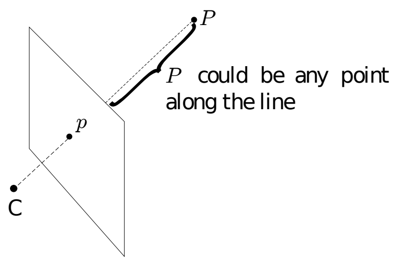


Figure 3.3: An image frame with a single 3D point P projected to a 2D point p. From the perspective of camera C, point P could lie anywhere on the line P.

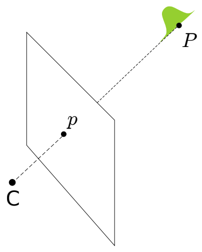


Figure 3.4: An image frame with 2D point p estimated to have an inverse depth with a Gaussian distributed probability at 3D location P.

tracked between the two frames, which corresponds to how visually similar the images are.

3.3.2 Depth Estimation

New image frames are used to update the depth map D_{kf} of the current key frame. This part of the process assumes that the relative pose estimation between frames is correct, but that the depth map is not. Once a new key frame is chosen, the depth map of the previous key frame is projected to the new key frame as initialisation of the new key frame's depth map.

To refine a key frame's depth map, depth estimates are obtained using variable baseline stereo techniques. Since the relative pose between a new frame and the current key frame is known, and assumed to be correct, a depth estimate can be made using the stereo depth estimation technique described in Chapter 2.2.2; however, because each key frame has multiple frames, with their respective poses associated with them, the algorithm may evaluate the depth of a pixel from multiple view points. This is advantageous because it not only alleviates the problem of quadratically increasing error over distance that is usually associated with fixed baseline stereo by allowing for more options of stereo comparisons, but also allows for the depth estimation of a pixel to be refined even further by multiple stereo comparisons. Figure 3.3 to 3.7 provides a summary of how this is achieved, as an extension of the typical stereo description made in Chapter 2.2.2 and adapted from the paper by Jack et al. [26].

Given a key frame with a depth map, assume a pixel p is given an inverse depth hypothesis d_p and variance σ_d, with a Gaussian distributed probability for the hypothesis as $\mathcal{N}(d_p, \sigma_d^2)$ as shown in Figure 3.4.

Figure 3.5 shows that with a new frame, and its given pose, the epipolar line can be calculated between the new frame and the key frame. From Figure 3.6, assuming the key frame has a depth hypothesis for pixel p, a search interval is

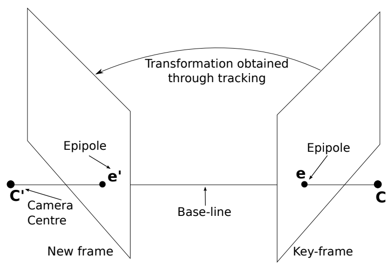


Figure 3.5: Epipolar constraint between camera centre C of the key frame and camera centre C' of the new frame, calculated using the new frame's pose estimation relative to the key frame.

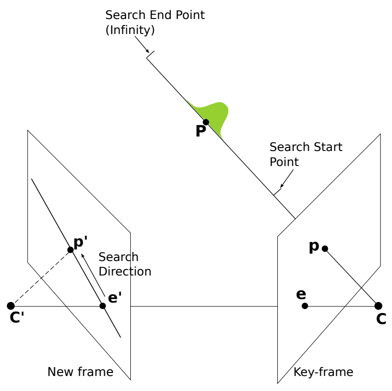


Figure 3.6: Point correspondence of pixel p, with a search along the epipolar line in the new frame, from the key frame.

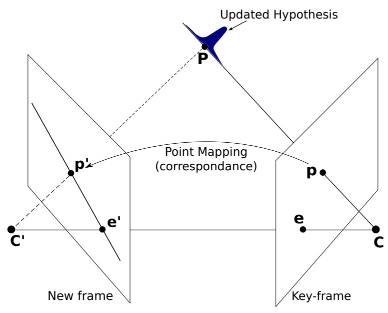


Figure 3.7: Successful point correspondence and depth estimation made using epipolar geometry to update the prior depth hypothesis.

designated for the point correspondence of the new frame to make the search more efficient. The disparity search interval is limited to $d_p \pm 2\sigma_d$, where d is the mean and σ_d is the standard deviation of the prior depth hypothesis. Figure 3.7 shows that a stereo comparison can be made by searching for pixel p along the epipolar line of the new image to determine the disparity for depth estimation. The existing depth hypothesis is then updated with the new depth estimate and the depth variance (which decreases to reflect a more accurate result). The variance calculation is based on the photometric (intensity gradient noise) and geometric (pose transformation noise) disparity errors [58]. If the depth has been estimated for a pixel which does not have a prior hypothesis, a new hypothesis is simply created with the estimated depth.

Once a new frame has been promoted into a key frame, the depth map estimates for the previous key frame are no longer updated. The final depth map is then regularised and outliers are removed [58]. The regularisation is done on every inverse depth measurement by averaging the surrounding inverse depth measurements, which are weighted by their respective inverse depth variances. Outliers are removed by keeping track of the validity of each inverse depth measurement, of every tracked pixel, by storing the probability that is is an outlier. If the pixel becomes occluded or if there are moving objects in the scene, it could cause a stereo search for the pixel to fail which causes the probability of it being an outlier to increase. Every successful stereo match decreases this probability. The depth map of a new key frame is initialised by re-projecting the final depth map of the previous key frame to the pose of the current key frame.

3.3.3 Map Optimisation

LSD-SLAM uses all finished key frames with their depth maps and pose estimates to create a point cloud map. The map is an accumulation of key frames joined by connecting edges to form a pose graph. Each key frame needs to be aligned correctly with one another to keep the map coherent and to adjust for translational, rotational and scale drift between key frames. This is done by finding similarity constraints between key frames, and optimising the pose estimation between them using a graph based back end library called *g2o* [68].

As mentioned before, an inherent problem with monocular vision is scale ambiguity, where absolute scale cannot be observed and hence scale drift occurs. This becomes evident when using monocular vision for larger maps with longer trajectories, which is why many monocular SLAM algorithms have been confined to smaller environments. LSD-SLAM attempts to solve this problem with a novel method called direct SIM(3) image alignment.

This method works very similarly to the image tracking method with Gauss-Newton minimisation as discussed in Section 3.3.1; however, instead of comparing a key frame with a new image frame, this method compares two finished key frames with their depth maps and pose estimates. As described in Section 3.2.3, the pose estimates in the image tracking stage only contained rotation and translation and therefore use pose $\xi \in \mathrm{SE}(3)$ as the representation. With the incorporation of scale, the pose representation becomes $\xi \in \mathrm{SIM}(3)$. Each key frame's depth map is scaled such that the mean inverse depth is equal to one. This ensures that the scale difference between key frames is incorporated into the pose estimation, SIM(3), which allows for explicit detection of drift in scale. By using direct image alignment on SIM(3) poses, two differently scaled key frames can be aligned. The new error function $E(\xi)$ describing the direct SIM(3) image alignment is shown in Equation 3.3.7 as a progression from the image tracking equation discussed in Section 3.3.1 as

$$E_p(\xi) = \sum_{p \in \Omega_{kf_1}} \left((I_{kf_1}(p) - I_{kf_2}(p'))^2 + ([p']_3 - D_{kf_2}(p'))^2 \right) \tag{3.3.7}$$

where

ξ is the 3D camera pose of key frame 1 relative to key frame 2 in SIM(3),

I_{kf_i} is the *ith* key frame image,

D_{kf_i} is the inverse depth map of key frame i,

$I_{kf_i}(p)$ is the relevant pixel of the key frame i,

$p' = \omega(p, D_{kf_i}(p), \xi)$ is the transformed pixel from key frame 2 to key frame 1 by ξ.

The error function as an extension of Equation 3.3.3 (preserving the definition of r_p and σ_{r_p} in Equation 3.3.4 and 3.3.5 respectively) then becomes

$$E_p(\xi_{ij}) = \sum_{p \in \Omega_{D_i}} \left\| \frac{r_p^2(p, \xi_{ij})}{\sigma_{r_p}^2(p, \xi_{ij})} + \frac{r_d^2(p, \xi_{ij})}{\sigma_{r_d}^2(p, \xi_{ij})} \right\|_\delta, \tag{3.3.8}$$

with

$$r_d(p, \xi_{ji}) := [p']_3 - D_j([p']_{1,2}), \tag{3.3.9}$$

and

$$\sigma_{r_d}^2(p, \xi_{ij}) := V_j([p']_{1,2}) \left(\frac{\partial r_p(p, \xi_{ij})}{\partial D_j([p']_{1,2})} \right)^2 + V_i(p) \left(\frac{\partial r_p(p, \xi_{ij})}{\partial D_i(p)} \right)^2 \tag{3.3.10}$$

where $p' := \omega(p, D_i(p), \xi_{ji})$ for a transformed point.
In Equation 3.3.8, r_p^2 and $\sigma_{r_p}^2$ is the photometric residual and variance respectively, while r_d^2 and $\sigma_{r_d}^2$ is the depth residual and variance respectively. The Huber norm is also applied to the sum of the residuals here to remove outliers because if one of the residuals is an outlier, the other is usually one too [4].

When a key frame is added to the pose graph map, loop closure possibilities are also tracked by collecting the ten closest key frames. An optional extra appearance based loop closure detection (OpenFABMap [59]) is used for large scale loop closures, as mentioned in Section 2.2.5. The final map is optimised using a *g2o* pose graph optimisation method with a resulting map shown in Figure 3.8.

3.4 Using LSD-SLAM

The authors of LSD-SLAM [4] report good results in RMS errors measured for absolute trajectories on various datasets. In their paper the tests, with documented results, were done on "small environment" datasets which consisted of scenes of office desks. For larger trajectories, tests have been performed by other authors [26; 67; 66] with differing results. However, the distinction between using either the open-source or commercial version (or possibly the

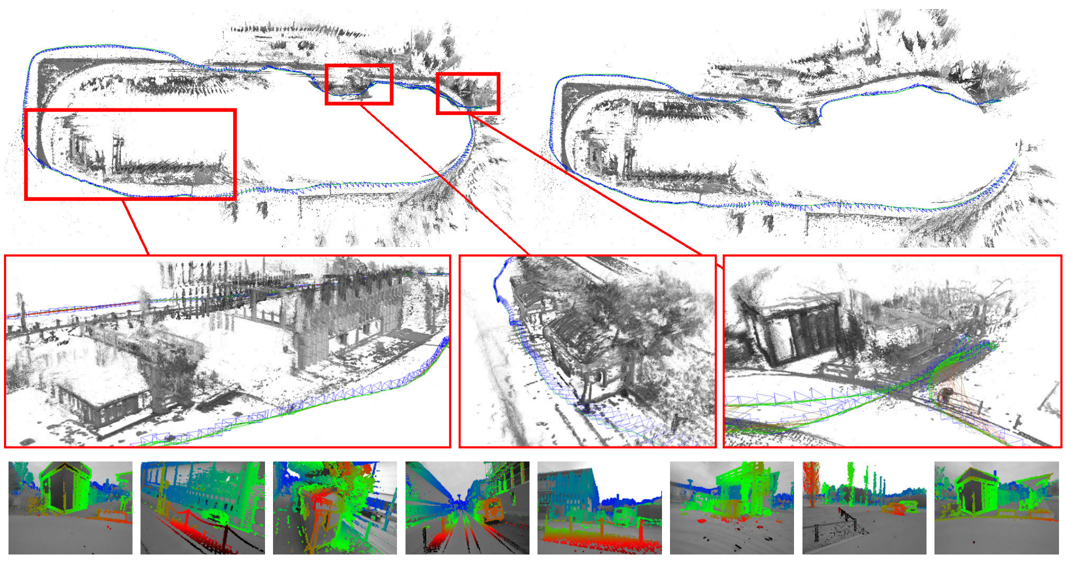


Figure 3.8: Large scale outdoor map created by LSD-SLAM. The top right map shows the result before loop closure, and the top left map shows the result after loop closure. The middle row of images show close ups of selected areas of the map, and the bottom row of images show selected key frames with their semi dense depth maps [4].

lab version) of LSD-SLAM has often not been clarified for these tests, with the exception of Jack et al. [26] who used the open-source version. The open-source version, which is available online [69], also makes it evident that many other users have experienced various issues with its implementation (for both dataset and live use) ranging from it not compiling, to not working at all, to extremely bad or incomparable results.

The official documentation and forum [69] for the LSD-SLAM project has some useful information regarding requirements to achieve better results:

Global Shutter - Using a rolling shutter on a camera may lead to poorer results.

Lens - A wide field of view lens is recommended. The authors of LSD-SLAM used a 130° fish-eye lens.

Frame Rate - A minimum frame-rate of 30 fps is required, unless the camera movement is very slow. In their paper they used between 30 and 60 fps.

Resolution - The recommended input resolution is 640 x 480 pixels. If the resolution differs significantly, then adaptations for this may need to be hard coded.

Initial Movement - At the initialisation of the input sequence it is recommended to move the camera in a circle parallel to the scene, without rotations. This may be to allow the initial random depth map to converge before continuing to the next key frame, although there is no reference to this assumption.

General Movement - LSD-SLAM does not do well with fast rotations, especially without sufficient translational movements at the same time. It is preferred to have the camera move sideways, forward or backward, or around the optical axis.

Parameter Adjustment - It is possible to adjust certain parameters of LSD-SLAM based on the dataset or scene for which the algorithm operates, such as camera pixel noise, minimum pixel gradient detection thresholds and the maximum distance allowable before key frame promotion. These parameters have a large effect on the resulting performance of LSD-SLAM on a given dataset.

Non-Deterministic - LSD-SLAM is largely non-deterministic in that results will differ when run on the same dataset twice, due to the parallelism and the "butterfly effect" that key frame selection has on the end result.

The above requirements for good results may be seen as restrictions for the algorithm, and must be taken into account when testing the resulting map. The datasets available specifically for LSD-SLAM work well; however, they do not have ground truth available and therefore one could only use these datasets to compare results with results from the LSD-SLAM paper.

Another two challenges are that LSD-SLAM cannot measure absolute scale, and the initial alignment of the map differs between the time the input sequence starts and ends due to the final optimisation step. This means that when comparing to a ground truth map, the resulting map would need to be rescaled and aligned to the ground truth map. Determining this scale and alignment poses a problem since *manual alignment* is prone to error.

Any other datasets used would need to be chosen to fit not only the requirements of acceptable results for LSD-SLAM, but also the requirements of this thesis. This would include:

Sufficient Texture - Since LSD-SLAM uses intensity gradients to identify pixels for tracking and depth estimation, it is important to use a dataset that has sufficient texture. Large blank or uniform walls may cause tracking failure because there are too few pixels to track. This can be subjective since if there is less texture in a scene, the thresholds for intensity gradients can be decreased to incorporate more sensitive detection of trackable pixels; however, if the scene changes to a more textured scene, then this would put considerably more load on the runtime of the algorithm when significantly more pixels are used. In other words LSD-SLAM would perform fast as a semi-dense algorithm until it is forced to build a dense map.

Ground Truth Map - Many datasets give ground truth in terms of pose trajectories. This is not completely useful when trying to evaluate the quality of the map itself. Therefore a dataset with ground truth of the environment is required for evaluation.

Loop Closure - To get a more accurate depth map from LSD-SLAM, the dataset should incorporate a loop in the trajectory so that it can optimise the map to correct for drift in scale. A map with drift may only prove useful for local navigation.

The examination and selection of datasets based on the above requirements are discussed further in Chapter 6.

3.5 Local, Large and Global Maps

This section will discuss maps in context of LSD-SLAM and in the context of the goals of this thesis. The definitions of maps will be kept as close to the norm as possible, however the details may be defined in a way that better suits the needs of the outcome for our system, whether they are properly defined in literature or not. Three types of maps will be discussed here: local, large and global maps. It is important to make the distinction between these maps because, as mentioned above, drift effects the result of each map which each have a separate purpose.

Local maps: Generally this refers to the immediate vicinity of the camera or robot. Some range is defined and any depth measurement within the defined range will be regarded as *local*. This is useful for obstacle detection or collision avoidance since there is no concern about colliding with an obstacle that is further away in a larger map. The exact range is variable and can be adjusted to the needs of the application or based on the environment. Our system has the option of creating such a map; however, it is not used for evaluation.

Large maps: A large map in the context of this thesis, refers to a continuous and persistent map of joint local maps, without optimisation. This means the resulting *large map* will still contain any drift errors that occur along the trajectory of the camera. Large maps may be useful to compare against global maps to visualise drift and corrections. In terms of navigation, a large map in this context may also be useful in the regard that *inaccurate data is better than no data*. The data may be incorrect if significant drift occurs and no loop closure is detected, but this map does need to be stored until such events occur any way.

Global map: Global maps represent the overall map of all pose and depth estimations, and is the main focus of this thesis. This representation does not necessarily include loop closure, and for the purposes of our goals, a global map is only created on either a loop closure or once the dataset has ended. All key frames with depth and pose estimations are joined into a single *large map*, and once a loop closure is detected and the large map is optimised, the entire *global map* is replaced. Both the local and large maps are updated after every new key frame is created, while the global map updates only on loop closures or when the data sequence has ended, although in the latter case the resulting global map may still contain significant drift depending on the performance of the mapping algorithm. The goal is to have a global map suitable for path planning or exploration, or even just accurate environment modelling.

Chapter 4

Occupancy Grid Maps

In the previous chapter we described the backbone of our system, LSD SLAM. The output of LSD SLAM is not immediately useful, as a point cloud, to an autonomously navigating system, and therefore needs further processing. In this chapter we will describe the goal representation of the resulting map for our system; occupancy grid maps. We will start with the definition of occupancy grid maps in Section 4.1, followed by the derivation of of the occupancy grid mapping algorithm, in Section 4.2, and an explanation of an important assumption made in Section 4.3. Section 4.4 deals with concepts to be taken into account when implementing occupancy grid maps, followed by an introduction to sensor models in Section 4.5.

4.1 Definition of Occupancy Grid Maps

Occupancy grid maps aim to divide the environment into an evenly sized grid of cells. The grid cells represent a portion of the environment and can store information corresponding to whether it is *occupied*, *free* or *unknown*. In the occupied case it signifies that there is an obstacle within the boundaries of the cell, and to a path planning algorithm or robot, this space is *not traversable*. In the free case, that portion of the environment *is traversable* and free of obstacles. While in the unknown case, there is no information about the state of the environment in that area, which may signify an area that needs to be explored. The grid can be divided into various resolutions depending on the detail required to represent the environment and can range, for example, from 1 cm to 5 m (resolutions outside this range are possible, but not typical). The grid cells in the map can each contain a value representing the probability of it being occupied, where the probability value ranges from 0 (free), to 1 (occupied), while a value of 0.5 would mean the state of the cell is unknown. This allows for sensor measurements, with noise or uncertainty, to be modelled into the map where for cell i:

m_i is the binary occupancy variable (occupied or free),

$p(m_i)$ is the probability of cell i being occupied with $1 = occupied$ and $0 = free$ and

$p(\bar{m}_i)$ is the compliment of $p(m_i)$.

The occupancy grid map can be structured as 2D or 3D depending on the requirements of the application. The 2D case may be useful for a ground-based mobile robot with a fixed height in an environment with fixed elevation, where only walls and doorways would need to be modelled for its 2D navigation. The 3D case may be useful for robots that move in 3 dimensions such as flying or submersible robots, or robotic manipulators in 3D space. In a 3D occupancy map, the cells are generally referred to as voxels. Figure 4.1 shows an example of an environment mapped into a 2D and 3D occupancy grid map.

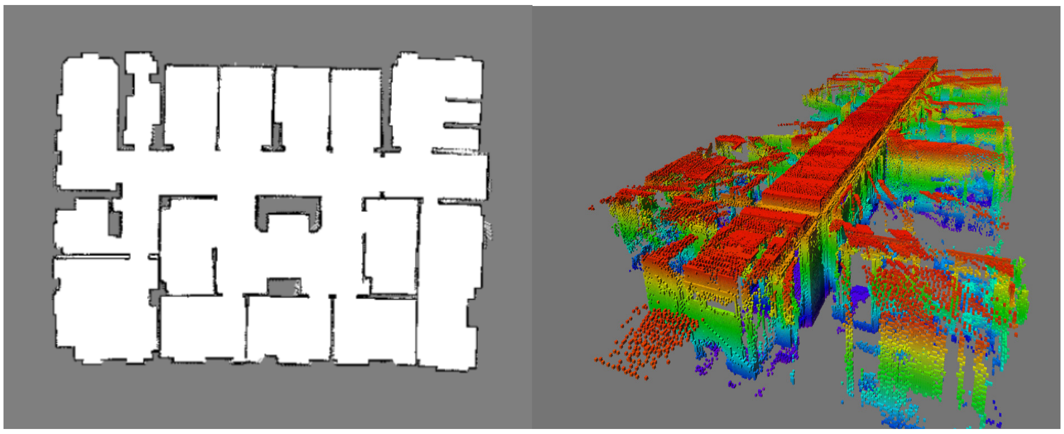

(a) 2D Occupancy Grid Map (b) 3D Occupancy Grid Map

Figure 4.1: A representation of a 2 Dimensional (a) occupancy grid map of an indoor office environment [9] and a 3 Dimensional occupancy grid map of the computer science campus at the University of Freiburg is shown in (b) [10].

At this point it is worth noting that the open-source library *OctoMap* [10] is used throughout this project. OctoMap was developed to fill a gap in the needs for an occupancy grid mapping library that provides solutions to the requirements of most robotic applications. According to Hornung et al. [10] these requirements are to represent a map probabilistically while modelling occupied, free and unknown space in an efficient manner in terms of memory usage and runtime. A summary of these requirements can be made as follows:

Probabilistic Representation - To create a map of the environment, sensor data is needed. Typically sensors are associated with some degree of error or uncertainty. For a range sensor, the error may be a few centimetres off from the actual range to an obstacle. This can be caused by sensor noise, reflections, dynamic scenes or other environmental factors. This uncertainty must be incorporated into the map probabilistically. Using a probabilistic approach also allows multiple uncertain measurements to be fused into a more accurate one that reflects the actual state of the environment. It also allows fusion of measurements from multiple sensors or even multiple robots.

Modelling of unknown space - While the importance of knowledge of occupied and free space is well appreciated for path planning robots, knowledge of unknown space is equally important. If there is uncertainty about the existence of a possible obstacle, then the area should be avoided in terms of it being used as a viable path. However unknown space can also help model parts of the map that are incomplete or unexplored.

Efficiency - For an autonomous robot to operate in real time, the map needs to be accessed efficiently in terms of speed, as well as be memory efficient. Not only does this effect the run time of the system, but also reduces power consumption and hardware performance requirements (and therefore cost). Having a map representation small in size, even with large environments, also allows for fast transmission of the map to multiple robots or servers.

In the following section, we will derive the occupancy grid mapping algorithm, on which OctoMap is based.

4.2 Algorithm Derivation

The following derivation is based on work popularised by Moravec et al. [35] with detailing by Thrun [11], Joubert [27], Burger [36] and Hornung [10]. In an ideal case, a system would calculate the posterior distribution over a map m given sensor measurements z and pose estimates x from beginning to time t (denoted by subscript $1:t$), which yields Equation 4.2.1 as follows:

$$p(m|z_{1:t}, x_{1:t}). \tag{4.2.1}$$

For convenience, by making the assumption that all sensor measurements z (that are relative to the pose x of of the robot) are transformed to world frame coordinates, we can eliminate x as the pose is then incorporated into the measurement z. The require posterior distribution map then becomes

$$p(m|z_{1:t}). \tag{4.2.2}$$

The map m is divided into equally sized grid cells with index i where a single cell in map m is represented as:

$$m = \{m_i\} \tag{4.2.3}$$

The probability of each grid cell m_i being occupied can be represented by $p(m_i = 1)$, while $p(m_i = 0) = 1 - p(m_i = 1)$ corresponds to the probability of the cell being free. Generally, because we can calculate the probability of a cell being free using the probability of occupancy, we shorten $p(m_i = 1)$ to $p(m_i)$ and $p(m_i = 0)$ to $p(\overline{m}_i)$ for convenience. The problem with calculating the posterior distribution over the map, as in Equation 4.2.2, is one of dimensionality. For a grid map of 1000 cells, the number of possible maps to represent becomes 2^{1000}, which is not a feasible task. It therefore makes sense to reduce the problem to individual grid cell evaluations as:

$$p(m_i|z_{1:t}) \tag{4.2.4}$$

To avoid this problem, we assume map cells are statistically independent of one another. In other words, it does not allow us to represent the dependencies of neighbouring cells but rather the posterior over maps as a product of its marginals as shown in Equation 4.2.5. This assumption has important implications and will be discussed further in Section 4.3.

$$p(m_1, m_2 \ldots, m_N|z_{1:t}) = \prod_{i}^{N} p(m_i|z_{1:t}) \tag{4.2.5}$$

The goal is to calculate the probability that cell i is occupied as part of the joint probability distribution of $p(m)$, given a set of measurements as $p(m_i|z_{1:t})$. The binary Bayes filter is well suited for this problem when considering a static environment with binary states, as well as the conditional independence between measurements, given m_i, over time. This is achieved, while applying Bayes' rule, as follows:

$$\begin{aligned} p(m_i|z_{1:t}) &= \tfrac{p(z_t|m_i, z_{1:t-1})p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})} \\ &= \tfrac{p(z_t|m_i)p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}. \end{aligned} \tag{4.2.6}$$

Applying Bayes' rule to $p(z_t|m_i)$:

$$p(z_t|m_i) = \frac{p(m_i|z_t)p(z_t)}{p(m_i)}. \tag{4.2.7}$$

Substituting Equation 4.2.7 into Equation 4.2.6 gives the probability of cell i being occupied as:

$$p(m_i|z_{1:t}) = \frac{p(m_i|z_t)p(z_t)p(m_i|z_{1:t-1})}{p(m_i)p(z_t|z_{1:t-1})}. \tag{4.2.8}$$

And similarly giving the complimentary probability of cell i being free as:

$$p(\bar{m}_i|z_{1:t}) = \frac{p(\bar{m}_i|z_t)p(z_t)p(\bar{m}_i|z_{1:t-1})}{p(\bar{m}_i)p(z_t|z_{1:t-1})} \tag{4.2.9}$$

To eliminate some tedious calculation we can divide Equation 4.2.8 by its compliment, Equation 4.2.9:

$$\frac{p(m_i|z_{1:t})}{p(\bar{m}_i|z_{1:t})} = \frac{p(m_i|z_t)}{p(\bar{m}_i|z_t)} \frac{p(m_i|z_{1:t-1})}{p(\bar{m}_i|z_{1:t-1})} \frac{p(\bar{m}_i)}{p(m_i)} \tag{4.2.10}$$

To avoid numerical instabilities for probabilities near 0 or 1, we can use log odds ratios to represent the occupancy as shown in Equation 4.2.11, although the probabilities can be easily recovered as shown in Equation 4.2.12.

$$L(m_i|z_{1:t}) = \log\left(\frac{p(m_i|z_{1:t})}{1 - p(m_i|z_{1:t})}\right) \tag{4.2.11}$$

$$p(m_i|z_{1:t}) = 1 - \left(\frac{1}{1 + e^{L(m_i|z_{1:t})}}\right) \tag{4.2.12}$$

By using log odds ratios, it also converts the multiplication of probabilities into addition of log odds values, which reduces computational expense and complexity. This can then be applied to Equation 4.2.10 to give Equation 4.2.13 as:

$$\log\left(\frac{p(m_i|z_{1:t})}{p(\bar{m}_i|z_{1:t})}\right) = \log\left(\frac{p(m_i|z_t)}{p(\bar{m}_i|z_t)}\right) + \log\left(\frac{p(m_i|z_{1:t-1})}{p(\bar{m}_i|z_{1:t-1})}\right) - \log\left(\frac{p(m_i)}{p(\bar{m}_i)}\right) \tag{4.2.13}$$

With simplified notation, as in Equation 4.2.11, this becomes:

$$L(m_i|z_{1:t}) = L(m_i|z_t) + L(m_i|z_{1:t-1}) - L(m_i) \tag{4.2.14}$$

Equation 4.2.14 represents the probability of a cell being occupied based on all previous measurements ($z_{1:t-1}$), given a new measurement (z_t). Following the three terms on the right hand side, this probability can be calculated incrementally using the newly acquired information, plus the previous information, minus the prior belief of occupancy. This is the basis of the *inverse sensor model*, which will be discussed in detail in Chapter 5, and is used to update the map by integrating new information of a cell's occupancy, without needing to update the entire map.

With the derivation above, and specifically Equation 4.2.11, the remainder of this thesis will assume that the probability of occupancy for a cell may be easily converted between probability and log odds ratios.

Equation 4.2.14 and Table 4.1 were derived with the assumption of a static environment. Based on the values in Table 4.1 and assuming that a free and occupied observation are equally likely for the sensor model, this means for k number of observations that a cell is free, approximately k number of observations of the cell being occupied also need to be made in order for the cell to be considered occupied, according to the occupancy threshold [10]. This is a desirable outcome for a static environment, however it poses problems

Table 4.1: Relationship between probabilities $p(m_i)$ and log odds ratios $L(m_i)$ for occupancy of cell i with the interpretation of each case.

$p(m_i)$	$L(m_i) = \log\left(\frac{p(m_i)}{1-p(m_i)}\right)$	Occupancy Description
0	$-\infty$	Definitely Free
0.5	0	Unknown
1	∞	Definitely Occupied

Table 4.2: Example of clamping thresholds being set with $l_{\text{max}} = 3.5$ and $l_{\text{min}} = -2$ with their corresponding probabilities and interpretations.

$p(m_i)$	$L(m_i) = \log\left(\frac{p(m_i)}{1-p(m_i)}\right)$	Occupancy Description
0.12	-2	Definitely Free
0.5	0	Unknown
0.97	3.5	Definitely Occupied

for a robot creating or using the map in a dynamic environment. This has lead to an extension for adaptability by [70], where a clamping policy can be employed to define an upper and lower bound for occupancy estimations. This is implemented in the OctoMap library as well, and can be described as:

$$L(m|z_{1:t}) = \max(\min(L(m_i|z_t) + L(m_i|z_{1:t-1}) - L(m_i)), l_{\text{max}}, l_{\text{min}}) \tag{4.2.15}$$

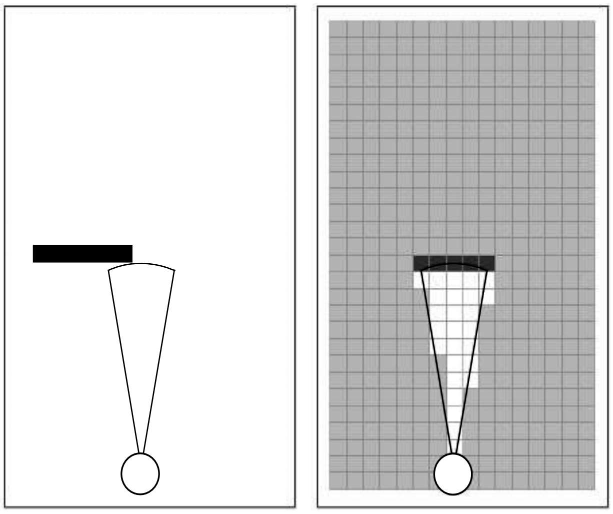
where l_{max} and l_{min} are the upper and lower bounds on the log odds update equation (4.2.14) respectively. This clamping policy provides two major advantages. Firstly it intuitively puts a limit on the number of updates needed to change the state of the cell. This implies that the confidence of the map remains bounded which allows the map to change quickly in accordance with a changing environment. Secondly, it allows for the compression of neighbouring cells with values that are close to occupancy bounds, with very minimal loss of information (depending on the clamping threshold). Of course, with this policy, any probability information close to 1 or 0 will be lost; however, probabilities between the clamping thresholds are preserved. An example of how the clamping policy is employed is shown in Table 4.2.

4.3 Conditional Independence Assumption

In the previous section we made the assumption that neighbouring cells in a map are statistically independent of one another. In other words, the state of a grid cell can be calculated without needing to consider the state of any other grid cells. This assumption was made to reduce the problem of high dimensionality, from estimating all possible maps for a large number of grid cells, to an estimate of the posterior over maps as a product of its marginals, as:

$$p(m_1, m_2 \ldots, m_N | z_{1:t}) = \prod_i^N p(m_i | z_{1:t}) \tag{4.3.1}$$

This assumption is consequential as there are cases where it may be invalid, such as when the measurement beam of a range sensor is wider than a grid cell, or when the range uncertainty spans multiple grid cells [36; 11]. In the first scenario where the measurement beam is wider than a grid cell, as is the case with typical sonar measurements shown in Figure 4.2, the sonar measurement returns a range value of an obstacle *somewhere* within the measurement beam. Since it cannot be sure where exactly within the width of the beam an obstacle is located, it simply considers all the cells in the beam width to be occupied at that range.



(a) Obstacle Location (b) Perceived Location

Figure 4.2: Sonar sensor partially sensing an obstacle (a), and returning the measurement as being *somewhere* within the measurement cone (b), where white cells are considered free, gray cells are unknown and black cells are considered occupied.

This causes a potential problem when making multiple measurements of a portion of the environment from different view points. As seen in Figure 4.3, a second measurement overlaps the first measurement. This creates a conflict in the assumption of whether a grid cell is occupied or not. Assuming all occupied grid cells from the first measurement are given a probability of 0.8, the probabilities should change based on the second measurement. Cells that are now in conflict should reduce in probability of occupancy (to 0.4 for example), while the originally occupied cells measured by the first measurement, which are out of view of the second measurement, should be increased in occupancy probability (to 0.95 for example). This is the basis of dependence between neighbouring grid cells. However, by implementing dependence between grid cells, it means every cell's state would be a function of every other cell in the map. If the map is large, occupancy calculations very quickly become infeasible due to the high dimensionality.

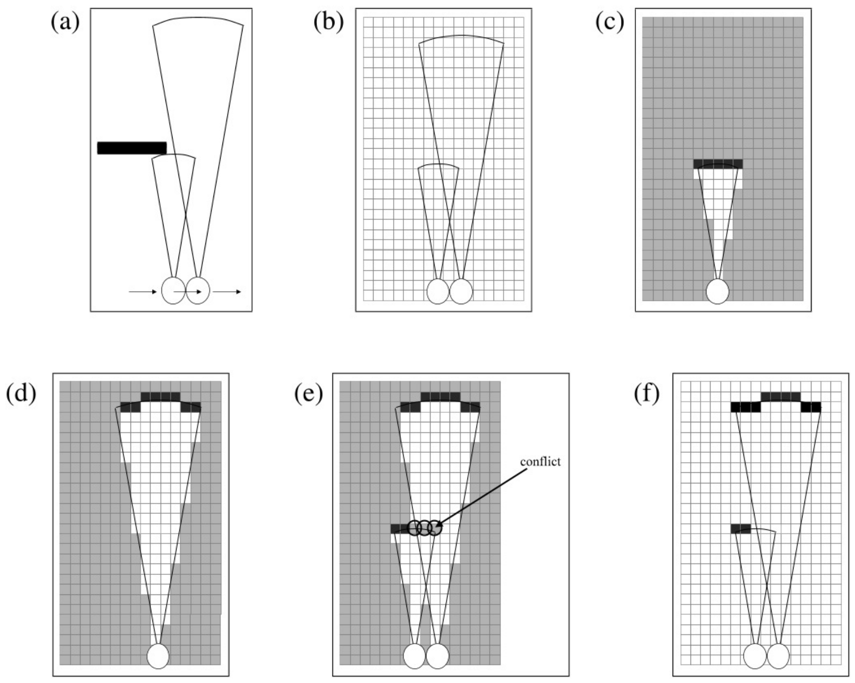


Figure 4.3: Multiple overlapping range measurements made with a sonar sensor (a) for occupancy grid map in (b). The first measurement detects an obstacle *somewhere* within the measurement beam and regards an obstacle to cover the width of the measurement cone (c). A second measurement is made, (d), where the maximum range is detected. Both measurements together cause a conflict as shown in (e), and with conditional dependence, the result should be resolved as shown in (f) [11].

In the second scenario for a range measurement with a small beam width but large range uncertainty, it can be shown (see Figure 4.4) that the assumption of independence can still be violated. Here, a range measurement suggests that an obstacle exists in one of 4 cells (Figure 4.4 a). White cells represent free space, a shade of gray represents a probability of the cell being occupied where light gray is a low probability and dark gray/black represents a high probability of occupancy. Figure 4.4 b shows that if a measurement is made from another view point, one of the cells could be considered free, meaning the remaining "occupied" cells should gain an increase in their probability of being occupied. Figure 4.4 c shows another measurement being made, clearing the second of the two cells, but updating the last remaining cell to a high probability of being occupied. The last remaining cell is never in direct view of the second or third measurements, and therefore shows how it is conditionally dependent on the state of the previous cells.

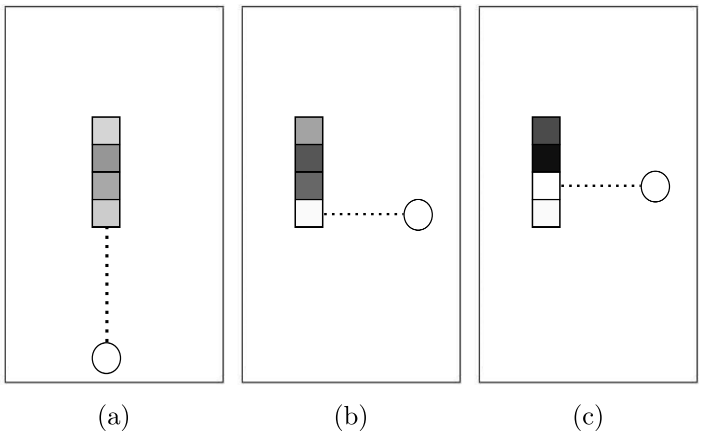


Figure 4.4: A sensor with a narrow beam width but large uncertainty measures an obstacle resulting in a probability of occupancy as indicated in (a). With each successive measurement made from a separate viewpoint, (b) and (c), the probability of occupancy increases for cells which were previously measured as occupied, while new measurements of old cells suggest they are free.

This thesis is concerned with vision sensors; specifically monocular vision, although monocular vision does employ stereo techniques when estimating depth of a scene. Generally for stereo vision measurements there is a large *range uncertainty* [71] which causes notable dependencies between grid cells. As mentioned in Section 3.3.2, LSD-SLAM reduces the effects of quadratically increasing error over distance, that is usually associated with fixed baseline stereo, due to the fact that there are simply more options for stereo comparisons to be made. Yet in LSD-SLAM there is still some uncertainty in range estimations, which is clear, due to each inverse depth estimate being associated with an inverse depth variance (a by-product of problems solved by optimisation). The assumption of independence would reduce the quality of the resulting map, however this assumption must still be made to make the problem tractable. This is especially crucial for maps that change in environment scale from small office scenes to large outdoor scenes. This implies that a map with a high resolution of a large environment would be used to calculate probabilities of all statistically dependent cells, resulting in an immense number of calculations. Considering the change in environmental scale and variable baseline stereo are key features of LSD-SLAM, it is therefore reasonable to make the assumption of conditional independence between grid cells.

This is advantageous because it not only alleviates the problem of quadratically increasing error over distance that is usually associated with fixed baseline stereo by allowing for more options of stereo comparisons, but also allows for the depth estimation of a pixel to be refined even further by multiple stereo comparisons.

4.4 Implementation

4.4.1 Measurement Beam

A range measurement beam typically consists of a range and a measurement angle. The measurement angle represents the outwards propagation of the transmitted signal over the range that is to be received by the sensor. The received signal could be from an obstacle anywhere from within this measurement width. For sonar the measurement angle is wider, while a laser's angle may be almost negligible. Figure 4.5 shows how the width of the measurement beam becomes modelled as a cone shape in the case for sonar sensors.

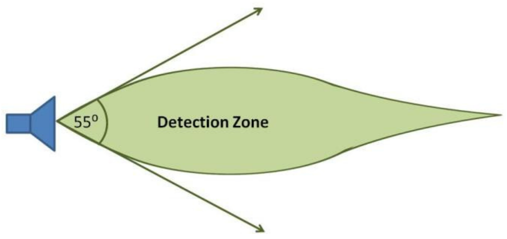


Figure 4.5: Basic illustration of the shape of the measurement beam of a typical sonar sensor. In this case the model represents the SRF05 Sonar Sensor (a commonly used ultra-sonic sensor) with the widest point of the beam being approximately 1 metre wide and a maximum distance of approximately 4 metres [12].

For vision sensors, it works slightly differently in that there is no transmitted signal, but only received light signals from an illuminated environment. Vision sensors like CCD cameras use pixels to represent a portion of the scene they are looking at. The further away obstacles are, the less number of pixels are used to represent the obstacle. In other words, an obstacle such as a pole may take up 30 pixels when viewed up close, but only 1 pixel when viewed from a distance. Depending on the camera parameters, the beam width for a vision sensor can be as small as 0.01°, which equates to an uncertainty of less than 10 mm at a distance of 5 m. Therefore it is reasonable to assume that a vision sensor's beam width to have a laser like ray structure instead of a cone shape. The assumption of a single ray per pixel greatly simplifies calculations and complexity for estimating the depth of each evaluated cell and also allows us to assume independence between cells more efficiently. It also allows for more efficient path tracing of the measurement beam through cells within the grid, also known as *ray casting*.

4.4.2 Ray Casting

Ray casting is an integral part of updating the cells in an occupancy grid map. Using Figure 4.6, we will describe the basics of ray casting in the context of a single measurement. Assume the pose of a sensor and its range measurement is known in world coordinates, as shown in Figure 4.6(a), where the measurement width is assumed to be negligible as a single ray. The range measurement is traced along a line between the pose and the obstacle, intersecting some grid cells along the way. The idea is to update each cell from the sensor to the measured obstacle with a value corresponding to its occupancy, as shown by Equation 4.2.14. Intuitively every cell between the sensor and the obstacle should be marked as free, while the cell in which the obstacle lies, should be marked as occupied.

Using an ideal sensor model (to update a map where all cells are initially unknown) as shown in 4.6(b), an iterative estimation of the probability of occupancy for each cell along the ray, can be applied. The ideal sensor model shows that for a range measurement of 1 m, the probability of occupancy is $p(m_i|z_t) = 1$. Every cell between the sensor and the actual range measurement has an occupancy probability of $p(m_i|z_t) = 0$, while every other cell that is not observed by the sensor has a probability $p(m_i|z_t) = 0.5$, which corresponds to the cell being unknown. The ideal sensor model implies that once the ray hits an obstacle, any cells further than the obstacle are not in view. This results in an updated grid map as shown in 4.6(c), where white represents free space, black represents occupied space, and gray is unknown space. An ideal sensor model implies perfect measurement data of the environment, which in reality is simply not possible. Sensor measurements of the environment are always associated with some degree of uncertainty, and this can be incorporated into an inverse sensor model to assign a characteristic probabilistic result to cells along a measurement beam. Incorporating measurement uncertainty into an inverse sensor model is described in further detail in Chapter 5.

There is a possibility of a ray missing an obstacle that only partially occupies a cell (see Figure 4.7). In this scenario, the cell may be falsely updated as free. One way to deal with this problem is to count the number of hits and misses for the cell, and then splitting the cell into a smaller cell (higher resolution) until the child nodes of the cell are considered either completely occupied or completely free, in a process known as adaptive grid mapping which is thoroughly described by Joubert [27] and Einhorn et al. [72]. Of course this requires multiple observations of the same cell in question in which case, given enough measurements, the cell would reach a clamping threshold and the problem would correct itself. Without using adaptive grid mapping or clamping thresholds, another method to handle partially occupied cells is to simply increase the resolution of the occupancy grid map. This may not always be a valid solution as it requires significantly more cells to be evaluated and could reduce the performance of the update algorithm, an effect which is

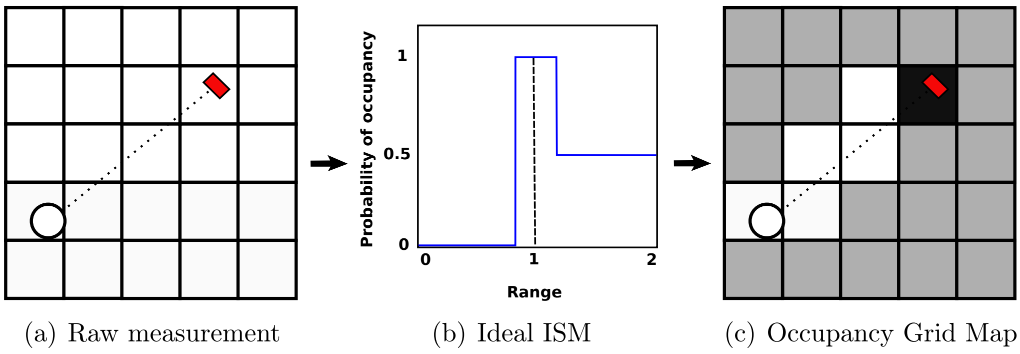


Figure 4.6: Example of implementation of ray casting in 2D, where a range measurement is made to an obstacle (shown in red) (a), and applied to an ideal sensor model (b) to update the grid cells of the map (c). Free cells are shown as white, occupied cells are shown as black and unknown cells are shown as gray.

highlighted in Chapter 6.

Even with the scenario of a partially occupied cell, with enough measurements, the neighbouring cell that contains a larger portion of the obstacle would be considered either completely occupied or unknown. In the occupied case, a path planning algorithm should allow for a minimum allowable distance from the edge of occupied space as a constraint for a possible path. In the unknown case, it should consider the region as “in need of further exploration”. Adaptive grid mapping is not within the scope of this thesis; however, we do employ upper and lower clamping thresholds, as discussed in Section 4.2.

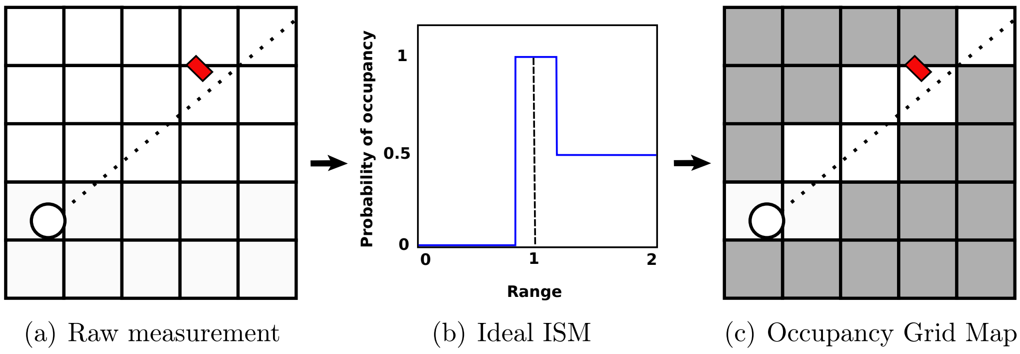


Figure 4.7: Example of implementation of ray casting in 2D, where a range measurement misses an obstacle that only partially occupies a cell (shown in red) (a), and applied to an ideal sensor model (b) to update the grid cells of the map (c). Free cells are shown as white, occupied cells are shown as black and unknown cells are shown as gray.

4.4.3 Pose Uncertainty

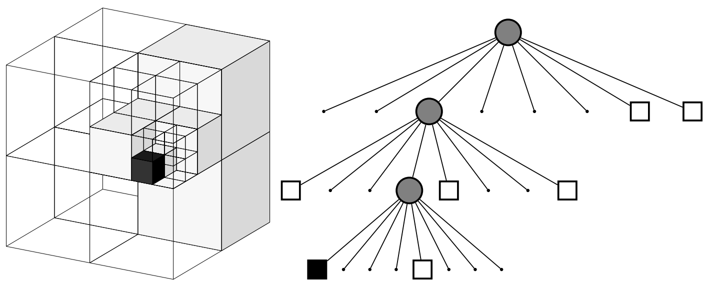
Updating the map using ray casting relies on an accurate estimate of the sensor or robot pose. In the real world, there is always some degree of uncertainty in the pose. The uncertainty can be modelled using a probability density function (PDF) as described by Thrun et al. [11], and can then also be incorporated into the measurement uncertainty [27] (incorporating measurement uncertainty alone by application of an inverse sensor model is discussed in detail in Chapter 5). Occupancy grid maps are mainly utilised in the post-processing phase. At this stage of an autonomously navigating system, the SLAM problem is assumed to have already been solved and the map is then updated or used for path estimation. By updating the map with a sensor model, it is assumed that either the pose is accurate, or that the pose uncertainty is incorporated into the the measurement uncertainty. In terms of LSD-SLAM, we regard the SLAM process as complete and take the pose estimates as correct, until a loop closure occurs as discussed in Section 3.3.3, at which point a new map is created with the newly updated pose estimates, which are also assumed to be correct in the context of a global map. Explicitly incorporating pose uncertainty into the resulting map is not possible using occupancy grid maps, unless it is incorporated with a sensor model, and is not within the scope of this project.

4.4.4 OctoMap

In Section 4.1 we briefly introduced the open-source OctoMap library and some of the solutions it attempts to solve in the mapping sector of robotics [10; 73]. Occupancy grid mapping is not a new concept and generally has a universal implementation across applications. Hence it makes sense to create a common framework off which to work from to develop occupancy grid applications. OctoMap is one such framework that allows occupancy grid mapping to be done efficiently in 3D.

The OctoMap framework is based on an octree hierarchical data structure to represent the environment in 3D. The main contributions of this framework are to allow for efficient probabilistic updates of free and occupied space while minimising memory consumption. The octree data structure, shown in Figure 4.8 for simplicity, consists of nodes that each represent a volume in 3 dimensional space, also known as a voxel. Each voxel can then be recursively sub-divided into 8 smaller volumes until a minimum voxel size is reached, which determines the resolution of the octree. The sub-divided volumes are known as children of the parent node, where every layer above a sub-division is a parent to that sub-division. The minimum node, or smallest child (resolution), is also known as a leaf node of the tree. The octree data structure also allows for efficient multi resolution queries to make use of inner nodes at a specific level.

Sensor measurements are integrated into the map at the finest resolution,



(a) Volumetric Representation (b) Tree Representation

Figure 4.8: Visual representation (left) of an octree voxel where free space (white) and occupied space (black) are stored in cells. The largest block represents the root node or entire map, and the small black block represents an occupied leaf node. Figure (a) shows the volumetric representation and (b) shows the tree representation.

or at the leaf node level, by application of an ideal sensor model and an update equation. Only the cell where the measurement occurs is updated as occupied, as shown in Figure 4.6. The update equation calculates the probability of a leaf node as:

$$\left(\frac{p(m_i|z_{1:t})}{1-p(m_i|z_{1:t})}\right) = \left(\frac{p(m_i|z_t)}{1-p(m_i|z_t)}\right)\left(\frac{p(m_i|z_{1:t-1})}{1-p(m_i|z_{1:t-1})}\right)\left(\frac{1-p(m_i)}{p(m_i)}\right) \tag{4.4.1}$$

Where z_t is the current measurement, $p(m_i)$ is the prior probability and $p(m_i|z_{1:t-1})$ is the previous estimate. With the assumption that a uniform prior probability for all cells is $p(m_i) = 0.5$, Equation 4.4.1 can be rewritten in the more efficient log-odds notation as:

$$L(m_i|z_{1:t}) = L(m_i|z_t) + L(m_i|z_{1:t-1}), \tag{4.4.2}$$

with

$$L(m_i) = \log\left[\frac{p(m_i)}{1-p(m_i)}\right], \tag{4.4.3}$$

where

$$L(m_1) = \log(1) = 0. \tag{4.4.4}$$

Octree can store maps in a significantly more compact manner by merging common leaf/child nodes into a single parent node. If all leaf/child nodes within a parent node contain the same log odds or binary value for occupancy, they are merged and represented by the single parent node with the same value. This is the basis for the notion of *adaptive grid size*, where the resolution of the grid size of a map adapts to the information within the map.

The OctoMap library comes with a 3D visualiser called *Octovis*, that can be used to visualise and manipulate the 3D occupancy grid maps with relative ease. Cell states can be manipulated with Octovis, and the entire map format can be changed as well. Maps formats are stored as either a binary tree, with *.bt* extension, or a full probability tree, with *.ot* extension. Binary trees store only two states (definitely free and definitely occupied), which greatly reduces memory consumption. The full probability tree stores all probabilities (up to a clamping threshold) of every cell, and is less memory efficient than a binary tree. All unknown or unmapped space is modelled by its explicit absence of information. Any 3D occupancy grid maps presented in this thesis are generated with Octovis.

An extension program that was made for this thesis, which is functionality not made available by OctoMap, was a visualisation for 2D views of 3D occupancy grid maps. The program was written to transform a 3D map and incrementally create an image slice for every step of a leaf node, spanning the entire height of the map. The program can also be used to display how the adaptive grid size is implemented in the map, and is also used for evaluating the map in Chapter 6. Some 3D maps may not be axis aligned or to the correct scale, and a program was written to perform this function as well. Any 2D occupancy grid maps presented in this thesis are generated using these programs. All tools that were developed to aid in this thesis are made freely available from `https://github.com/GrimHull/Octomap-Tools`.

4.5 Forward Sensor Model

The basic implementation of occupancy grid maps, and as described above for Octomap, is to implement measurement data using the ideal sensor model. However, it is far more beneficial to incorporate measurement uncertainty into the map, and this can be achieved using a *forward* or *inverse* sensor model. The *forward sensor model* could be considered the most intuitive way of thinking about using sensor measurements to create a map of the environment. It aims to calculate the probability distribution of sensor measurements z_t given a map m_i as:

$$p(z_t|m_i). \tag{4.5.1}$$

Thrun [28] uses the forward sensor model, and the expected maximisation (EM) technique, to calculate maps that are most likely to cause the full set of sensor measurements up to time t. The problem is approached without the independence assumption between cells, which comes with the cost of high dimensionality. The high dimensionality of the problem relates to the computational cost of estimating the map, as every time a new measurement is made, a new map must be calculated. This is not ideal for situations where a robot exploring an environment needs to create a map incrementally while still using the continuously updated map at the same time. Incremental estimates of a map using forward sensor models have been done by Pathak et al.[74], where estimated probabilities of visibility are assumed to directly determine probabilities of occupancy, although the resulting performance is shown to be inferior to other sensor models that determine occupancy probabilities directly. Forward sensor models are often determined experimentally. Along with the non-incremental nature and the fact that occupancies are only assigned full probabilities, as either definitely occupied or definitely free, we will not utilise forward sensor models in this thesis. Instead, the *inverse sensor model* will be employed, which is given as:

$$p(m_i|z_t). \tag{4.5.2}$$

In the following chapter, we will discuss and derive an inverse sensor model to create a map based on sensor measurements. We will also discuss an alternative inverse sensor model to later compare with our derived model.

Chapter 5

Inverse Sensor Model

Inverse sensor models (ISMs) were first introduced by Elfes [75] in 1989, and later detailed by Thrun [11]. ISMs aim to use sensor measurements to assign a probability distribution to a portion of an occupancy grid map that represents free or occupied space. From the view point of the sensor, all cells between it and the obstacle that is measured need to be populated with a probability value representing the likelihood for a binary state of free or occupied. Any cells beyond the obstacle, or cells that have not been viewed by the sensor, can be regarded as unknown. The probabilistic state assignment of a cell helps to incorporate noise and uncertainty into the map, which is an important consideration to make as sensor measurements are generally corrupted by some form of noise and uncertainty. Therefore when designing an ISM, sensor noise should be taken into account. The general equation for an ISM (as mentioned in Section 4.2) is:

$$p(m_i|z_{1:t}) \tag{5.0.1}$$

The ISM is required to update the occupancy states of a map, and can be derived in various ways. This chapter will focus on two popular ISMs specifically for the output of LSD-SLAM. First, some assumptions will be made, and goals will be set out to form specifications for our sensor model. In Section 5.2 an ISM will be derived mathematically, followed by a discussion of parameters, in Section 5.2.3, that may be tweaked to obtain different results. The derivation is based on work done by Thrun [28] and Joubert [27], with some adaptations made for the purposes of this project. Another ISM proposed by Andert [76] will be described in Section 5.3, also followed by a discussion of parameters that change the resulting output, in Section 5.3.3. Finally, Section 5.5 will briefly describe inverse sensor models for other sensors.

5.1 Specifications and Assumptions

To derive a sensor model, some specifications and important assumptions need to be outlined. Specifications regarding what type of measurements we receive and how to treat the measurements, shape the overall model of the sensor.

5.1.1 Specifications

We start with the measurement information that is received from the LSD-SLAM algorithm. As input to our sensor model, LSD-SLAM provides a 2 dimensional array of inverse depth values per key frame, each with an associated inverse depth variance. The 2D array has a size matching the resolution of the camera. If the camera has a resolution of 640 x 480 pixels, then the 2D inverse depth array will be this size too. However, not all elements will contain depth information, as not all pixels in the image are used for depth estimation. These disregarded pixels are then simply assigned a inverse depth value of 0, which will be ignored in the implementation of the sensor model. All inverse depth estimates are simply inverted to get the correct depth range value.

LSD-SLAM also provides a pose estimate for every key frame containing completed inverse depth information. The pose is represented in SIM(3) format (as discussed in Section 3.2.3) and is used to transform each key frame depth estimate (where a key frame pose is relative to the previous key frame pose) to world frame coordinates, and also used to *draw* a ray from the camera origin to the measurement end point.

Every inverse range measurement made by LSD-SLAM is associated with uncertainty, which is assumed to be Gaussian distributed [4], and this must be taken into account to create a reliable map. The uncertainty of a measurement can be incorporated into an update of the map by adding independent Gaussian noise to the measurement, modelled by the associated uncertainty.

The output of the ISM will be a probability value for each cell along the ray. The probability value will be converted to a log odds ratio to make cell updates easier and less computationally intensive. This has large implications for speed as there are potentially thousands of cells to evaluate per ray for each key frame, and our system must still keep up with real time performance.

5.1.2 Assumptions

Some assumptions need to be made that effect how the sensor is modelled to keep it tractable.

We start with the assumption of independence, where (as discussed in Section 4.3) every cell's probability of occupancy is independent from every other cell in the map. This is aided by the assumption that every depth measurement made by the sensor is structured as a ray, originating from the key frame origin, with a negligible beam width. This allows the sensor model to be modelled as a 1 dimensional evaluation along the measurement ray to the measured range. For each key frame, each depth value z larger than zero with its variance σ_z, will be evaluated individually.

The uncertainty is also assumed to be normally distributed around the inverse depth measurement. Although this is rarely the case in reality, it is common practice as it simplifies calculations and yields satisfactory results [30] [28] [75]. Using Gaussian distributions to represent noise of a measurement allows for efficient calculations as it is a uni-model function with one single maximum. This proves to be effective when modelling a measurement with a relatively small margin of uncertainty where only a single distinct hypothesis exists [11].

5.2 Inverse Sensor Model with Gaussian Noise

5.2.1 Overview

The basic strategy of the following derivation for the first of two ISMs is to begin with an ideal sensor model, and convolve it with a Gaussian distribution function, as described by Thrun [28] and Joubert [27]. The ideal sensor model, shown in Figure 5.1, is modelled on the premise of a noiseless, perfect, sensor. Hard probabilistic assignments $p(m_i)$ are made to each cell i as being either definitely free ($p(m_i) = 0$), definitely occupied ($p(m_i) = 1$), or unknown ($p(m_i) = 0.5$). In this ideal case every cell before the obstacle or range measurement is set to *free* and every cell after the obstacle is set to *unknown*. At the actual range measurement of the obstacle, a buffer range L is set to account for the possibility that the measurement is not centred on a cell within the grid. The distance to the sensor is calculated using the centre of the cell where the measurement is made. Ideally, a measurement made anywhere within this cell will assign a 1 to this cell. Generally the size of band L around the measurement range is chosen to be the diagonal distance between corners of a cell [27]. However, as will be shown later in Section 5.2.3 and shown in our results (see Section 6.2.4), this choice may not always be intuitive or valid.

The Gaussian probability distribution function (PDF) of a noisy measurement, as shown in Figure 5.2, is used to model the noise of a sensor measurement. The variance σ_z indicates the uncertainty of a measurement z, which is represented by the width of the normally distributed curve. A larger variance indicates more uncertainty about a measurement and results in a wider curve. Conversely a smaller variance indicates less uncertainty about a measurement and a thinner curve.

The next step is to convolve the ideal sensor model with the normally distributed Gaussian PDF. Figure 5.3 shows a visual representation of what the convolution results in. The resulting model describes how an evaluation

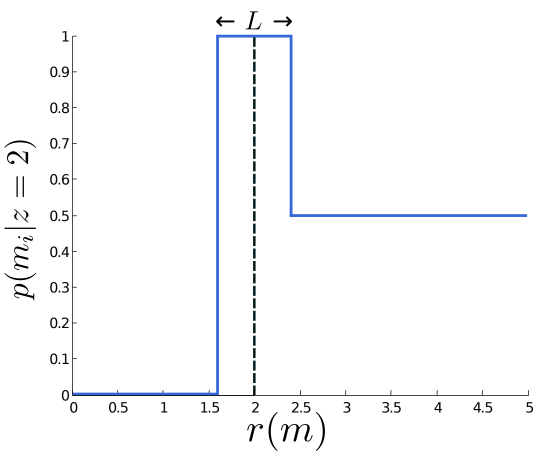


Figure 5.1: Ideal inverse sensor model with a range measurement of $z = 2$ m, and a buffer range of $L = 1$ around the measurement z. This model is used by Thrun as the basis of his derivation of his ISM.

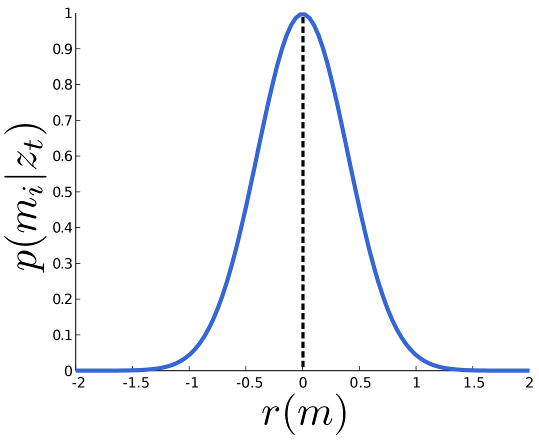


Figure 5.2: Gaussian probability density function with normally distributed uncertainty $\sigma_z = 0.5$.

of the range along a measurement should tend to 0 until it gets closer to the actual range measurement of the obstacle. The uncertainty creates a gradual rise in the probabilistic value that would tend to 1 for cells closer to the peak of where the actual measurement is believed to be, and then gradually drop towards 0.5 for cells behind the measurement range.

5.2.2 Analytical Derivation

In this section we will analytically derive an ISM by convolving the ideal sensor model, defined here as $g(r)$, with the Gaussian PDF, defined here as $f(r)$. The ideal sensor model as shown in Figure 5.1 can be represented as:

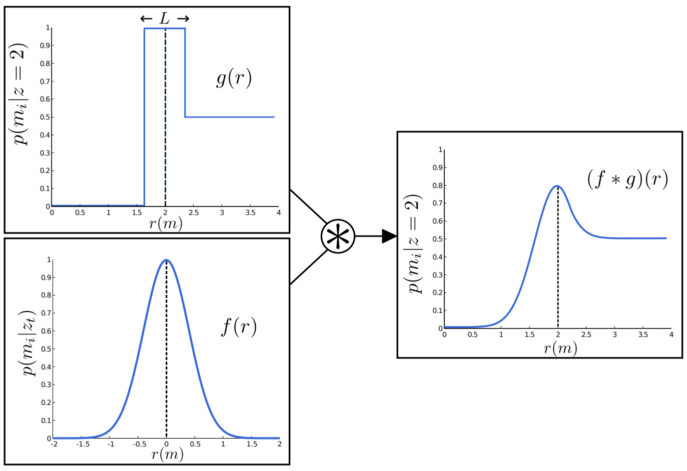


Figure 5.3: Convolution of an ideal sensor model (top left), with measurement of $z = 2$ m, and a Gaussian noise model (bottom left), with $\sigma_z = 0.3$, resulting in the Gaussian inverse sensor model (right), with measurement $z = 2$ m.

$$g(r) = \begin{cases} 0, & \text{if } r < z - \frac{L}{2} \\ 1, & \text{if } z - \frac{L}{2} \leq r < z + \frac{L}{2} \\ 0.5, & \text{otherwise for } r > 0 \end{cases} \tag{5.2.1}$$

Where realistically r must be positive since the sensor should never return a measurement made behind it. However, for convenience we will accept a negative range, $r < 0$, for the derivation of the sensor model. Recall that the range measurement z contains normally distributed Gaussian noise, with a standard deviation σ from z, which can be written as:

$$r \sim \mathcal{N}(0, \sigma), \tag{5.2.2}$$

The standard PDF of the Gaussian distribution as shown in Figure 5.2, where r is the range along the measurement line to the measurement z, is then:

$$f(r) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(r-0)^2}{2\sigma^2}} \tag{5.2.3}$$

The convolution formula for $f(r)$ and $g(r)$ can be made, as shown in Equation 5.2.4, to incorporate Gaussian noise into the ideal sensor model. However, the ideal sensor model must be handled as the piecewise form in which it is defined. The convolution can be calculated numerically, although to handle variable resolutions of the map grid sizes, an analytical method is preferable. For Equation 5.2.4 to be well defined, f or g must decay rapidly close to the end endpoints, which is accomplished by the Gaussian PDF.

$$(f * g)(r) = \int_{-\infty}^{\infty} g(\tau) f(r - \tau) d\tau \tag{5.2.4}$$

Since $g(r)$ is piecewise, as mentioned above, the convolution must be evaluated in a piecewise fashion implying that the integration must be split whenever $g(r)$ changes case. The complete convolution equation can be written as:

$$(f * g)(r) = kF(a, b), \tag{5.2.5}$$

where the interval (a, b), on which r lies, defines the value of $k \in \{0, 0.5, 1\}$. In other words the scaling factor k is chosen based on where range value r is located within intervals determined by Equation 5.2.1.

For simplification, Equation 5.2.6 is computed, where a and b would be the intervals on which the measurement z lies along range r:

$$F(a, b) = \frac{1}{\sigma\sqrt{2\pi}} \int_a^b e^{\frac{-(r-\tau)^2}{2\sigma^2}} d\tau. \tag{5.2.6}$$

Let :

$$u = \frac{r - \tau}{\sigma\sqrt{2}} \tag{5.2.7}$$

Equation 5.2.6 with the substitution of u then becomes Equation 5.2.8:

$$F(a, b) = \frac{-1}{\sqrt{\pi}} \int_{\frac{r-a}{\sqrt{2\sigma^2}}}^{\frac{r-b}{\sqrt{2\sigma^2}}} e^{-u^2} du. \tag{5.2.8}$$

This can then be represented by use of the error function as:

$$F(a, b) = -\frac{1}{2}\text{erf}\Big(\frac{r - b}{\sqrt{2\sigma^2}}\Big) + \frac{1}{2}\text{erf}\Big(\frac{r - a}{\sqrt{2\sigma^2}}\Big), \tag{5.2.9}$$

where the error function $\text{erf}(x)$ is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{5.2.10}$$

As an added convenience to the derivation, we calculate that if $a = -\infty$ the definition of $F(a, b)$ then becomes:

$$\begin{aligned} F(-\infty, b) &= \lim_{a \to \infty} \tfrac{1}{\sigma\sqrt{2\pi}} \textstyle\int_a^b e^{\frac{(r-\tau)^2}{2\sigma^2}} d\tau \\ &= -\tfrac{1}{2}\text{erf}\Big(\tfrac{r-b}{\sqrt{2\sigma^2}}\Big) - \tfrac{1}{2} \end{aligned} \tag{5.2.11}$$

Convolution of Piecewise Integrals:

Function $g(r)$ only changes value for the first time when $r = z - \frac{L}{2}$, therefore the range for $r < z - \frac{L}{2}$ is represented by $r \in (-\infty, z - \frac{L}{2})$ and is equal to 0.

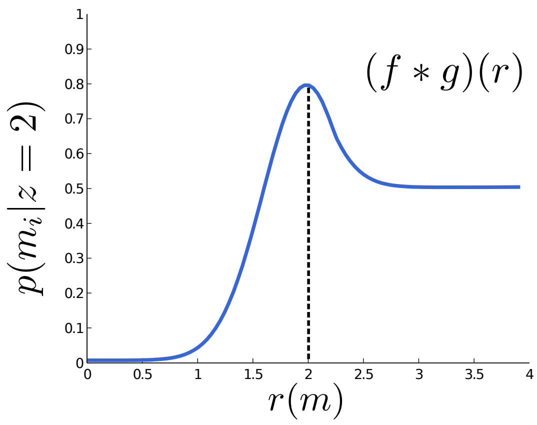


Figure 5.4: Derived inverse sensor model by convolution of the ideal sensor model and a Gaussian distribution function. Here the resulting probability is shown for a measurement $Z = 2$, with a mean range $r = 2$ and a depth variance $\sigma = 0.3$.

The second range where $g(r)$ changes value is at the peak of the measurement z within the buffer value L (which is centred around z), which is represented by $r \in (z - \frac{L}{2}, z + \frac{L}{2})$ and is equal to 1.

For the last interval where $g(r)$ occurs after the measurement, as $r > (z + \frac{L}{2})$, where the space becomes unknown and the probability tends to 0.5 for $r \to \infty$, represented as $r \in (z + \frac{L}{2}, \infty)$, is equal to 0.5. The complete piecewise convolution is shown by Equation 5.2.12 as:

$$\begin{aligned}
(f * g)(r) &= 0F(-\infty, z - \tfrac{L}{2}) + 1F(z - \tfrac{L}{2}, z + \tfrac{L}{2}) + 0.5F(z + \tfrac{L}{2}, r) \\
&= -\tfrac{1}{2}\text{erf}\Big(\tfrac{r-z-\frac{L}{2}}{\sigma\sqrt{2}}\Big) + \tfrac{1}{2}\text{erf}\Big(\tfrac{r-z+\frac{L}{2}}{\sigma\sqrt{2}}\Big) - \tfrac{1}{4}\text{erf}\Big(\tfrac{-z}{\sigma\sqrt{2}}\Big) + \tfrac{1}{4}\text{erf}\Big(\tfrac{r-z-\frac{L}{2}}{\sigma\sqrt{2}}\Big) \\
&= -\tfrac{1}{4}\text{erf}\Big(\tfrac{r-z-\frac{L}{2}}{\sigma\sqrt{2}}\Big) + \tfrac{1}{2}\text{erf}\Big(\tfrac{r-z+\frac{L}{2}}{\sigma\sqrt{2}}\Big) - \tfrac{1}{4}\text{erf}\Big(\tfrac{-z}{\sigma\sqrt{2}}\Big)
\end{aligned} \tag{5.2.12}$$

Equation 5.2.12 shows the final derived inverse sensor model formula. It represents the probabilistic value that is calculated given a depth measurement with a depth variance, exactly as shown in Figure 5.4

5.2.3 Parameter Discussion

In this section, we will discuss some of the parameters of our derived model, and how they alter the resulting probability calculations and shape of the curve.

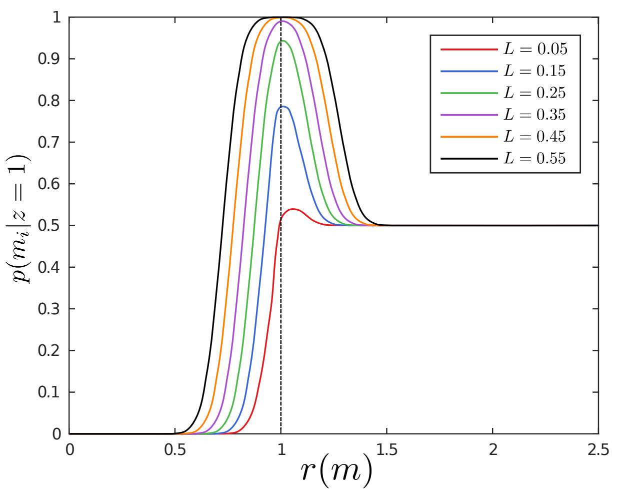


Figure 5.5: Effect of varying L values for Thrun's inverse sensor model at a range measurement of $z = 1$ m and a standard deviation $\sigma = 0.1$.

5.2.3.1 L value

Figure 5.5 shows how a varying buffer value L either narrows or widens the peak of the curve. A larger L value widens and increases the height of the peak, and a smaller value narrows and lowers it. As mentioned in Section 5.2.1, the value for L is generally chosen as the diagonal width of a cell in an occupancy grid map. If this value is made too large, it defines occupancy for more cells around the actual measurement along the measurement ray. In Section 6.2.4 we show the effect this parameter has on the accuracy of the resulting map and conclude if this method of selection is sufficient.

5.2.3.2 Standard Deviation σ

The standard deviation σ represents the uncertainty, of a prediction of occupancy, when applied to an ISM. Visually, this corresponds to a higher and narrower peak for a smaller σ, or a lower and wider peak for a larger σ, as shown in Figure 5.6. For a large uncertainty in a depth estimate, more cells around the actual measurement will likely be considered occupied. Each depth estimate calculated by LSD-SLAM, has an associated depth variance that is used to calculate the standard deviation of an occupancy prediction.

5.2.3.3 Range Dependent Uncertainty

As discussed above, LSD-SLAM does not suffer from range dependent uncertainty to the same degree as normal fixed baseline stereo, and we therefore do not consider depth dependent uncertainty in our final model; however, for interest, in Figure 5.7 we show how Thrun's ISM calculates probabilities of measurements made over a distance. From Figure 5.7 we see that Thrun's

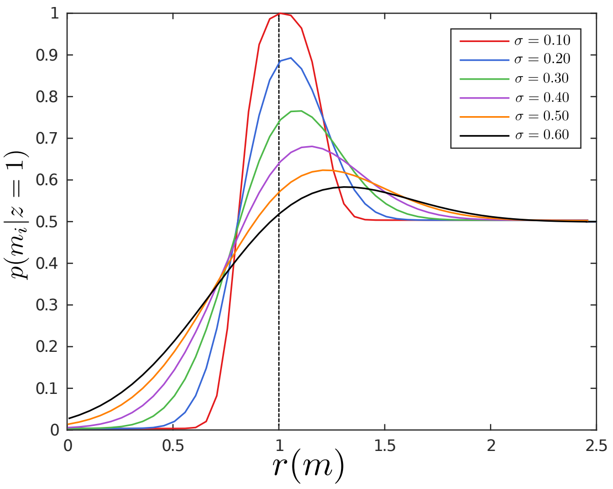


Figure 5.6: Effect of varying σ values for Thrun's inverse sensor model at a range measurement of $z = 1$ m and $L = 0.4$.

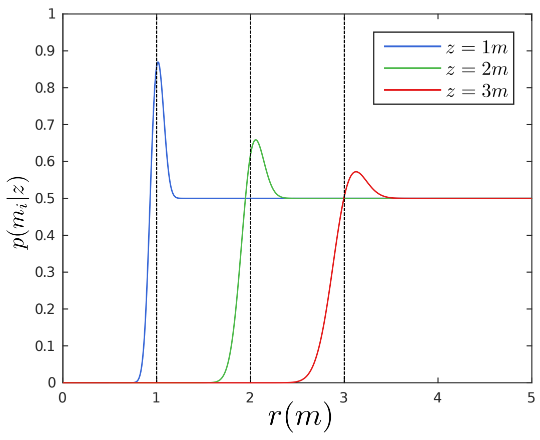


Figure 5.7: Effect of varying distance of measurements for Thrun's inverse sensor model assuming distance dependent uncertainty was taken into account, where measurements made at $z = 1$ m, 2 m and 3 m are made, with $L = 0.4$.

ISM seems to overshoot on the range where the peak probability on the prediction of occupancy located. The range at which the measurement is located is assigned a slightly lower probability than a range slightly further than the actual measurement. This would likely cause cells that actually contain an obstacle to be assigned a lower probability than the peak probability of the total measurement range.

5.3 Andert's Inverse Sensor Model

5.3.1 Overview

The second inverse sensor model we will consider, is one published by Andert [76]. Similarly to Thrun's method derived above, it softens hard predictions of occupancy made by an ideal sensor model to incorporate measurement uncertainty. Andert uses an exponentially quadratic component to transition between free and unknown predictions. Although Andert's ISM is originally designed for stereo vision, it can be easily modified to work for the monocular case. The following section will describe the model along with the adaptation from stereo to monocular sensors, and then a discussion on the parameters of the model is given in Section 5.3.3.

5.3.2 Algorithm

At the time of writing, details on the derivation of Andert's ISM have yet to be published. Therefore a description of the various components of the model will be given rather than an analytical derivation as given for Thrun's ISM in Section 5.2.2.

As a preliminary step, when considering the use of Andert's ISM for vision sensors, the resulting 3D location represented by pixel p, in camera coordinates, is described as:

$$p = (x_c, y_c, z_c)^T \tag{5.3.1}$$

A line can be drawn between the camera centre and pixel p with distance l_p, as shown in Equation 5.3.2.

$$l_p = \sqrt{{x_c}^2 + {y_c}^2 + {z_c}^2} \tag{5.3.2}$$

with the uncertainty of distance l_p being depth dependent and described as:

$$\Delta l_p = \Delta z_c \frac{l_p}{z_c} \tag{5.3.3}$$

Where z_c is the distance measurement in camera coordinates calculated from stereo disparity values as:

$$z_c = \frac{bf}{d} \tag{5.3.4}$$

Where:

$b =$ horizontal baseline distance between two stereo cameras

$f =$ focal length of the cameras

$d =$ disparity between corresponding pixels

As mentioned, Andert's sensor model was originally designed for fixed baseline stereo vision, which has the inherent problem of resulting in distance measurements with uncertainty that increases quadratically with distance, hence the depth dependent uncertainty in Equation 5.3.3. However, recall that variable baseline stereo vision is not effected by this problem in the same way fixed baseline stereo is, and is one of the key features implemented for monocular vision in LSD-SLAM. While the quadratic increase in error over distance still exists, variable baseline stereo allows for many more possible images to be used for stereo comparisons, which can reduce the error significantly. Therefore a small modification to Andert's method will be made here to better suit our system. The uncertainty of distance measurement shown in Equation 5.3.3, will simply be replaced with the uncertainty of the inverse depth measurement made by LSD-SLAM.

At this point we have the structure in place to calculate the distance to a 3D point in space, and draw a line to it from the camera centre. The aim is to iterate over every occupancy grid cell along the line and evaluate the probability of occupancy until the final range measurement is reached. The next step is to describe the sensor model. Similarly to Thrun's ISM, Andert's ISM starts with an ideal sensor model, as shown in Equation 5.3.5

$$P_{occ}(r) = \begin{cases} p_{min}, & \text{if } r \leq z \\ 0.5, & \text{if } r > z \end{cases} \tag{5.3.5}$$

where p_{min} is the minimum probability value that is assigned to a cell.

Andert's ISM attempts to minimise the effects of uncertainty Δl_p by incorporating a scaled quadratic exponential function with the estimated distance measurement z shown by Equation 5.3.6 as:

$$e^{-\frac{1}{2}\left(\frac{r-l_p}{\Delta l_p}\right)^2}. \tag{5.3.6}$$

Similar to the function of a Gaussian noise convolution with an ideal sensor model, the quadratic exponential function tries to cause the curve of a resulting probability to increase symmetrically around the measurement range and have a width related to the uncertainty of the measurement. For Andert's method, this is applied to stereo vision sensors and therefore the uncertainty increases with distance. The exponential function is scaled using Equation 5.3.7 as

$$\frac{k}{\Delta l_p \sqrt{2\pi}} + 0.5 - P_{occ}(r), \tag{5.3.7}$$

where significance factor k is used to specify the weighting that a single measurement has on the map. The significance factor ensures that measurements made further away have less impact on the map, while closer measurements have a higher impact. The combination of the significance factor, the

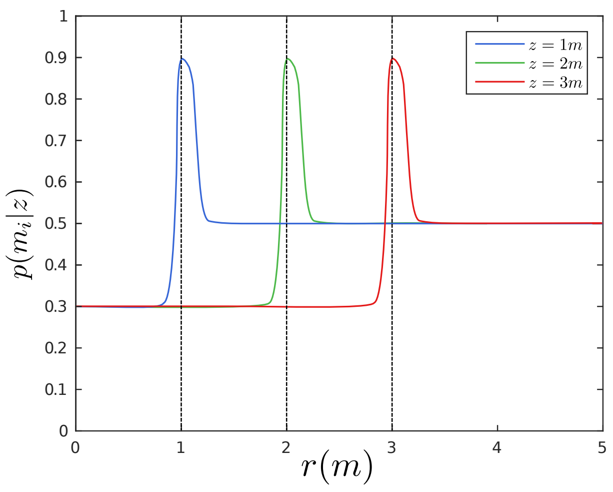


Figure 5.8: Inverse sensor model by Andert without depth dependent uncertainty, for a cell size of 0.1 m, standard deviation $\sigma = 0.1$ and significance factor $k = 0.1$.

scaling and the quadratic exponential function create an asymmetrical probability curve around a distance measurement, that gradually tends towards 0.5 after the range measurement is reached, or if maximum range is reached. Given that the measurement uncertainty is no longer depth dependent, the probability curve will stay consistent across all ranges.

The final ISM, with our adaptation to *monocular vision*, is then given by Equation 5.3.8 as

$$P(s(l)) = P_{occ}(l) + \left(\frac{k}{\sigma_{z_t}\sqrt{2\pi}} + 0.5 - P_{occ}(l)\right)e^{-\frac{1}{2}\left(\frac{r-z_t}{\sigma_{z_t}}\right)^2}. \tag{5.3.8}$$

The probability of a cell's occupancy for a *stereo vision system* is also determined with Equation 5.3.8, with the difference that σ_{z_t} is simply replaced with range dependent uncertainty Δl_p. Figure 5.8 shows the resulting probabilities given various range measurements without range dependent uncertainty. Every cell along the measurement range r is evaluated and assigned a probability of being occupied using Equation 5.3.8. For cells before the measurement, the probability assignment is the minimum to classify the cells as free space. For cells near to or on the measurement, the probability of occupancy increases in relation to the uncertainty of the measurement. Cells after the measurement range tend toward 0.5 to signify them as unknown.

5.3.3 Parameter Discussion

This section will take a closer look at the effects of key parameters of Andert's inverse sensor model, specifically the significance factor k and measurement

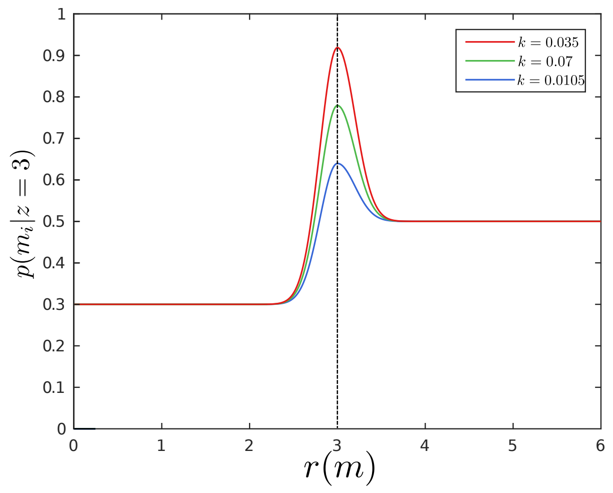


Figure 5.9: Inverse sensor model by Andert with varying significance factor k, with a cell size of 0.1 m, standard deviation $\sigma = 0.1$ and range measurement $z = 3$ m.

uncertainty σ. We will also show what the resulting probability curves would look like when considering depth dependent uncertainty.

5.3.3.1 Significance Factor k

Figure 5.9 shows the effects that the significance factor k plays on resulting probabilities based on depth measurements. The significance factor acts as a scaling factor for measurements. This is more applicable to stereo sensors to allow closer measurements (since they are more accurate) to have a greater impact on the map than those that are far away. How this factor is chosen is unclear, and therefore we still need to find a significance factor that would give the best results for all measurements that are made for a monocular system. This is done in Section 6.2.4 where we evaluate the effects that the significant factor has on the accuracy of the resulting map.

5.3.3.2 Standard Deviation σ

The standard deviation, σ, behaves very similarly to that for Thrun's ISM, as expected. Once again, as shown in Figure 5.10, higher uncertainty results in lower and wider peaks, while lower uncertainty results in thinner and taller peaks; however, Andert's ISM has a distinctively narrower curve making it more specific on its prediction of occupancy than Thrun's ISM. As with Thrun's ISM, the depth variance from LSD-SLAM is used to calculate the standard deviation of each measurement. When applying an inverse depth measurement to an ISM in our system, we use the standard deviation of the range estimate as returned by LSD-SLAM.

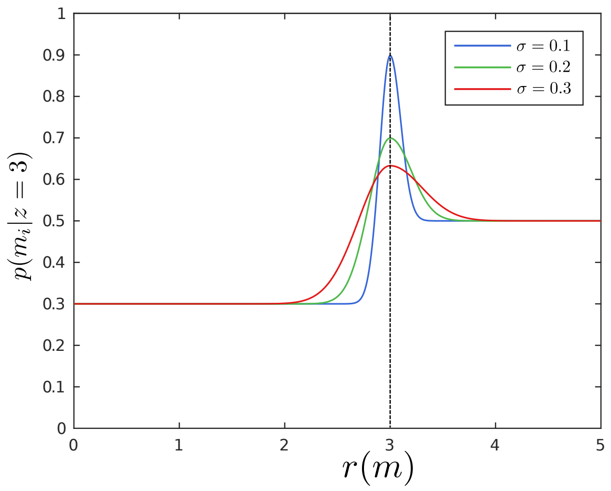


Figure 5.10: Inverse sensor model by Andert with varying standard deviation σ, with a cell size of 0.1 m, significance factor $k = 0.1$ and range measurement $z = 3$ m.

5.3.3.3 Distance Dependent Uncertainty

If Andert's ISM were to be applied to a stereo sensor, the representation of depth dependent uncertainty is shown in Figure 5.11. Closer measurements have a higher probability of being occupied, and the probability decreases with distance for each measurement until a maximum distance is reached, at which point the cell's status' become unknown. Also note how Andert's ISM makes a peak prediction of occupancy at the actual measurement range, in contrast to Thrun's ISM that makes a prediction of occupancy slightly after the actual measurement.

5.4 Log Odds Update for ISMs

At this stage we have discussed both ISMs by Thrun and Andert. The ISM's are used to determine the probability of occupancy for each cell along a measurement ray. The probabilities that are calculated, then need to be inserted into the actual occupancy grid map. Recall, from Section 4.2, that probabilistic updates are performed in terms of the corresponding log odds ratio. This is done to avoid numerical instabilities near 0 and 1, and also makes computation significantly more efficient by using addition of log odds versus multiplication of probabilities. This is simple to apply to the inverse sensor models proposed above as each cell along a measurement ray is evaluated individually. The probabilities are calculated and converted to log odds and added to the existing log odds value stored in the cell (if one exists).

Clamping thresholds are also set to cap the log odds values that are stored

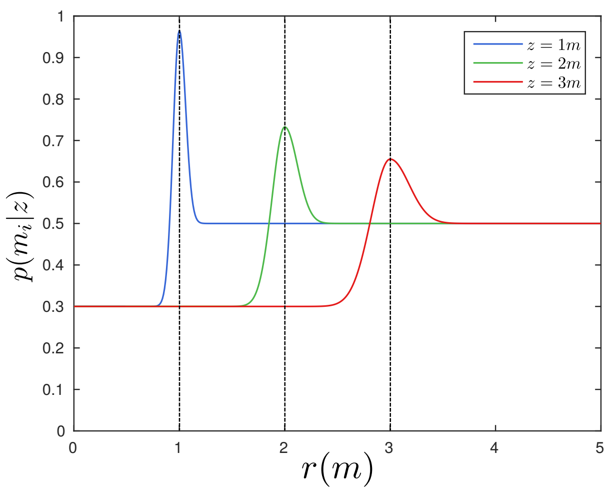


Figure 5.11: Inverse sensor model by Andert with depth dependent uncertainty, with a cell size of 0.1 m, significance factor $k = 0.1$. In this case the camera parameters are a baseline $b = 0.3$ m, focal length $f = 700$ pixels and a disparity uncertainty of $d = 0.5$.

in a cell. As shown in Table 4.2, the maximum log odds value stored for an occupied cell may be clamped, for example, at 3.5, and −2 for an unoccupied cell. These values correspond to probability values of 0.97 and 0.12 respectively. A cell being exactly 1 for occupied or exactly 0 for unoccupied is generally never applied in practice as we can never assume perfect knowledge of the environment. We can only be extremely confident, with a small degree of uncertainty, which may correspond, for example, to probability values of 0.99 for occupied and 0.01 for free cells, if we correctly meet all of our assumptions.

5.5 Inverse Sensor Models for Other Sensors

As demonstrated thus far, an inverse sensor model can be tailored to a specific sensor. This is often necessary because different sensors have different characteristics associated with how measurements are made as well as how they should be interpreted. For example, sonar sensors differ from lidar in that a laser is a single coherent beam and SONAR, using sound waves, has a cone structured measurement beam. This needs to be taken into account when developing a sensor model because an obstacle detected with a sonar sensor would yield a larger range of possible locations than that of a lidar sensor.

5.6 Conclusion

In this chapter we discussed in detail two different inverse sensor models; one by Thrun and one by Andert. These ISMs were designed differently and shown to be adaptable to specific requirements based on their application and the type of sensor used. In Chapter 6 we aim to compare these two sensor models to determine which may be best suited for autonomous navigation using LSD-SLAM's monocular vision depth data to build an occupancy grid map. Tests will be performed on each model by varying the resolution of the map, and a parameter of each sensor model. For Thrun's ISM we will vary the buffer size L and for Andert's ISM we will vary the significance factor k. These two parameters are chosen as there is very little information in literature on the decision for their values. For Thrun's sensor model the L value is said to be chosen so as to account for a measurement not centred on a cell. However, as shown in Section 5.2.3.1 the effect of changing this value has significant effects on the resulting occupancy probabilities on and around the measurement made. Likewise, for Andert's ISM, the introduction of the significance factor k is not associated with much more information for deciding how to choose its value. It too has a significant effect on the resulting map by having a varied effect on the updates of occupancy for a cell within the map, as shown in Section 5.3.3.1.

Chapter 6

Experiments

In this chapter we discuss some experiments that are performed on various datasets to compare two inverse sensor models described in the previous chapter. First, in Section 6.1, a description of the overall system is given, followed by a discussion on some important information of datasets for our evaluation, where we outline some restrictions that are imposed on our tests as a consequence. We then describe what we aim to measure and why, along with the methodology on how the experiments are performed. Finally in Section 6.2, the results of the tests are presented and discussed.

6.1 Part 1 - Methodology

As stated in Chapter 1, the aim of this thesis is to evaluate the performance and quality of a 3D occupancy grid map created using LSD-SLAM with monocular vision in real time. This means there are some key factors, that play a role in determining the viability of using such a map building system for autonomous navigation, that need to be tested. These factors are *memory consumption*, *computation time* and *map accuracy*. It is important to note that we are not evaluating LSD-SLAM itself, but evaluating our mapping system in conjunction with LSD-SLAM. All tests are therefore done on the resulting occupancy grid maps and the algorithms that create them rather than LSD-SLAM on its own. Specifically, we are testing and comparing two sensor models as outlined in Chapter 5, *Andert's* and *Thrun's* ISMs.

6.1.1 Project Overview

Figure 6.1 shows an overview of our system in conjunction with LSD-SLAM. A single camera is used to obtain a set of consecutive image frames of an environment. The image frames are processed into select key frames with associated depth information, as described in Chapter 3. The completed key frames are sent to our mapping algorithm that applies an ISM, either the

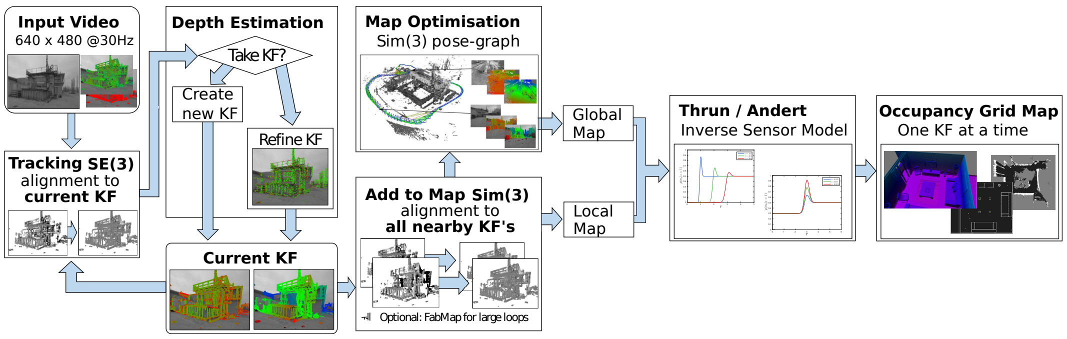


Figure 6.1: Overview of the complete system used to created occupancy grid maps.

derived model by *Thrun* or one by *Andert*, to create an occupancy grid map containing a probabilistic value of occupancy for each cell. The entire process is done on an image sequence in real time on an Intel i7-4700MQ CPU (8 cores at 2.40GHz each) with 16 GB of RAM. LSD-SLAM uses 2 cores (one for localisation and one for mapping), and our occupancy grid mapping algorithm runs on its own core. The advantage of running on its own core is that it runs in parallel with LSD-SLAM without significantly decreasing the performance of either LSD-SLAM or the occupancy grid mapping process.

The complete system runs on the Robotic Operating System (ROS) [44], which works on the principle of sending messages between nodes. LSD-SLAM contains two nodes, one for localisation and one for mapping, which corresponds to a core each, as a node uses one core of the processor. LSD-SLAM sends a completed key frame with depth information out as a message, and the message is received by our occupancy grid mapping node which continuously listens for incoming key frame messages. The key frame message is processed and added to a map while new key frames that are received are stored in a buffer until the previous key frame is completely processed. Two maps can be created; a local map and a global map. The local map is an accumulated concatenation of key frames regardless of loop closure, and therefore contains any drift that occurs in the depth estimation of LSD-SLAM. The global map is created once a loop closure is detected and is rebuilt from scratch upon every new loop closure within a certain number of key frames. Both maps may be stored in RAM during the run-time of the mapping process, and also saved to file in real time. We will be focusing on the global map, as it would contain the most accurate information of the environment due to the optimisation of the map upon completion of the dataset, as well as the possibility of correcting for drift through loop closure. Evaluating our system on the basis of a local map would be to evaluate the drift that occurs within LSD-SLAM, which is out of the scope of this research.

6.1.2 Dataset Discussion

Before experiments are done, some critical commentary needs to be made on datasets that are available for testing. Many datasets were tested to determine their usefulness for this thesis and descriptions of these tests, be it success or failure, are given in this section. These initial tests help determine how evaluation can be done and what obstacles need to be overcome to gain specific information, since some datasets do not contain ground truth, and those that do may not offer it in occupancy grid map form. It also helps to highlight the limitations associated with LSD-SLAM.

The tests we aim to perform in this chapter are ones on *memory, performance* and *map accuracy* (the latter of which is the main focus of this thesis). For our purposes, memory tests do not need to be concerned with speed or accuracy of the system. We are only interested in the on-board RAM usage and therefore ground truth maps are not necessary, while large and small datasets are necessary. For evaluation of performance, we are not concerned with accuracy or memory consumption, but purely on the speed at which map creation occurs, and if it is sustainable for real time applications. Therefore we only require datasets representing large and small environments, and no ground truth. Accuracy evaluation, on the other hand, requires that we compare our resulting map with a known ground truth of the map. This means datasets for these tests need to be accompanied with ground truth data of the environment. Regardless of what experiments need to be performed, the base requirement for any dataset used is that it must be purely visual (a sequence of image frames) as LSD-SLAM is a purely visual SLAM algorithm. Along with this requirement, it is also necessary for a camera calibration file to be associated with the dataset.

Another requirement that is sufficient, but not necessary, is that the dataset contains a loop closure. This would help mitigate issues with drifts in datasets that may skew the resulting map. This is only a sufficient condition in that a dataset may prove to be easy for LSD-SLAM to track between frames resulting in very little drift.

The authors of LSD-SLAM demonstrated successful use of 4 datasets, which are made available online [25]. The datasets vary in real life scenery from an office room, to an outdoor industrial setting, to a large outdoor food court and a medium sized "special" indoor scene (see Figure 6.2 for sample images for the first 3 datasets). The datasets are purely visual and are made available as a package of image sequences. There is no ground truth of the environment although the datasets are accompanied by point cloud depth maps of the scenes that were achieved by the authors using LSD-SLAM. The datasets successfully run and complete using the *open-source* version of LSD-SLAM, and although the results are comparable to theirs, our results are unsurprisingly not as good, even with altered parameters. These datasets can be used for evaluation on memory usage and performance, but purely on a qualitative basis in terms of accuracy. Quantitative evaluation of the map cannot be done without a ground truth version of the map. Note that this is not the evaluation of the set of poses or the trajectory. A benchmarking tool is also offered online to evaluate resulting trajectories in comparison to the results of other authors. However, our aim is to evaluate a resulting map of our system and sensor models, not the accuracy of the trajectory of LSD-SLAM itself. With this in mind, a dataset with a ground truth map is required.

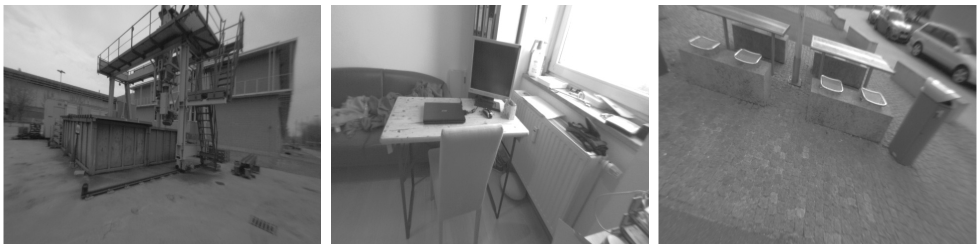

(a) Machine (b) PC Room (c) Food Court

Figure 6.2: Sample images of datasets used by the authors of LSD-SLAM [4]

TUM does make many real-life visual RGB-D SLAM datasets available online (see Figure 6.3 for some sample images), with ground truths of trajectories recorded using a motion capture system, and point clouds of the scenes that are built using a Microsoft Kinect [77]. The datasets vary in environment size and difficulty with regards to varying the speed and magnitude of the translation and rotational movements to cater for many evaluation needs. We tested some of the more simple datasets on LSD-SLAM, one of which consisted of fairly slow purely translational movements above an office desk environment. This dataset, and 9 others from the selection failed to either maintain tracking or yield meaningful results. We found that if LSD-SLAM did succeed in maintaining tracking, it was because of altered parameters, however the results were still unusable as there was no clear representation of the environment, in other words the depth estimation failed. This may have been caused by many factors including fast motion, low frame rates, too much rotation or even poor initial movements to converge to a stable depth estimate. The latter possibility may be considered the most likely as the authors of LSD-SLAM make this one of the requirements to getting the best result using their system.

Another dataset provided by Wasenmüller et al., called CoRBS (comprehensive RGB-D benchmark for SLAM), is available online [13]. It consists of 4 models (3 of which are shown in Figure 6.4) within a scene that contain videos or a sequences of image frames of trajectories around the models, and is accompanied with ground truth data for both the trajectories and the actual 3D models. The trajectories were measured with a motion capture system and the models were scanned with a 3D scanner, both with sub-millimetre accuracy. These datasets were run on LSD-SLAM and while LSD-SLAM did not fail in

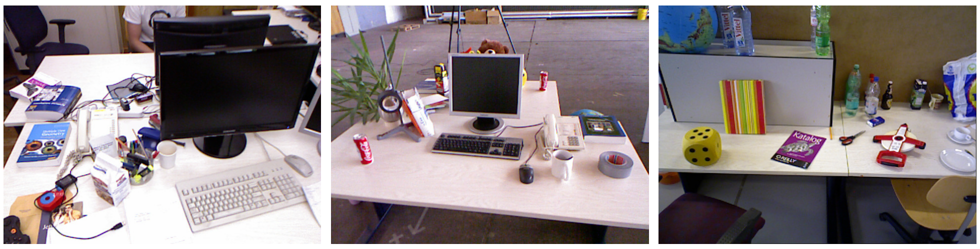

(a) Freiburg1 xyz (b) Freiburg2 desk (c) Freiburg3 long office

Figure 6.3: Sample images of more datasets offered by TUM

tracking, the results were unusable. The motion of the trajectories contained either too much rotation for LSD-SLAM to handle, or the the depth wasn't able to converge sufficiently.

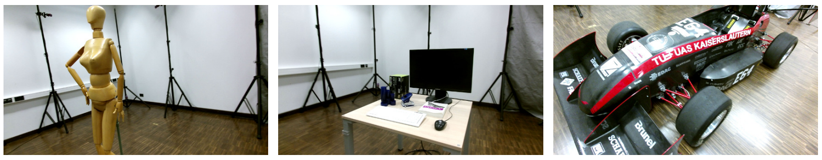

(a) Human (b) Desk (c) Racing Car

Figure 6.4: Sample images of the CoRBS datasets offered by [13]

Stanford University provided a dataset, now managed by *Zhou et al.* and also available online [14], that contains various scenes with video and image sequences of the trajectories, as well as ground truths for both trajectories and the actual scene. The scenes are of various environments (as seen in Figure 6.5) which include a copy room, a lounge, a reading room and many more. The scenes were scanned with a Microsoft Kinect and the resulting point clouds of the scenes are also made available. LSD-SLAM was able to initialise on these datasets; however, the tracking very quickly failed and the algorithm aborted. The best result achieved was only the first few seconds of the *copyroom* dataset although it was, however, too small of a result to be useful.

The Imperial College of London and the National University of Ireland Maynooth provides a dataset for benchmarking RBG-D, visual odometry and SLAM algorithms, known as the ICL-NUIM Dataset, created by Davison et al. [15]. This dataset is based on two synthetic scenes, a *living room* and an *office* scene, as shown in Figure 6.6. Four trajectories are available for each scene as a sequence of image frames for each trajectory, along with the ground truth of the trajectories. While the dataset is of a synthetic scene, an image sequence with added noise for each trajectory is also made available. One of the trajectories for each scene also completes a loop to the degree that the final image frame ends on with a view of the scene that was previously visible

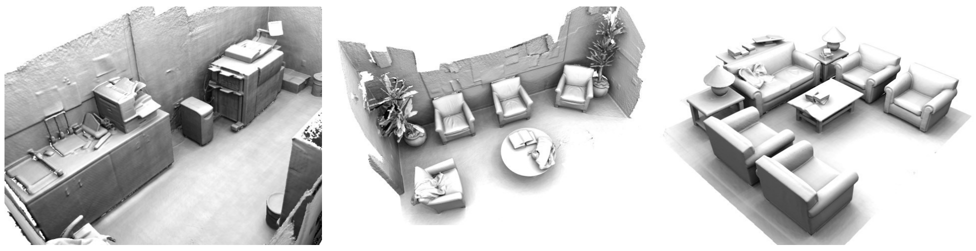

(a) Copy Room (b) Lounge (c) Reading Room

Figure 6.5: Sample images of the *Standford 3D Scene* datasets offered by Zhou [14]

in the first few frames. The loop does not continue further than that. The *living room* scene is the only one that has the ground truth model of the scene, available in the form of a point cloud. A ground truth point cloud of the office scene is available; however, the ground truth map is different to the actual scene represented in the dataset and is therefore not useful. All trajectories for both scenes, using *perfect* and noisy images, were tested on LSD-SLAM although only one trajectory for each scene completed the entire image set consistently. However, the results were not consistent between trajectories of the office scene and the living room scene. For the successfully completed runs, explicit loop detection failed in both cases, even though the resulting map did maintain enough scale information to be arguably good enough for the purposes of testing. With initial tests of the Living Room Dataset, it proved to be the best candidate for *quantitative* analysis since it fulfilled the requirements for these tests. Being the only dataset available that worked consistently on LSD-SLAM and offered a ground truth version of the map and trajectories, it allows for accurate evaluation of the resulting map. Evaluation using synthetic data is arguably far more accurate than datasets that provide ground truth maps that are constructed with a Microsoft Kinect, since there will always be inherent uncertainty in any physical range sensor. A counter argument is that the magnitude of uncertainty may be negligible when considering an environment on a very large scale, which is what LSD-SLAM is well suited for. While a large scale environment with a ground truth map is preferable, at the time of writing, one that works for LSD-SLAM could not be found.

The Living Room Dataset contained 4 different trajectories, one of which completed without tracking failure or inconsistent results, and maintaining scale information throughout. This trajectory will be used for quantitative evaluation of our system. More datasets would be preferred, however the literature at the time of writing showed that there are simply not enough candidates for visual based SLAM datasets (specifically monocular vision) with explicit ground truth maps of both small or large scale environment.

The above discussion of available datasets highlights some limitations for this project in that only one dataset meets the requirements of *quantitative* testing. For experiments on *memory consumption* and *performance*, where

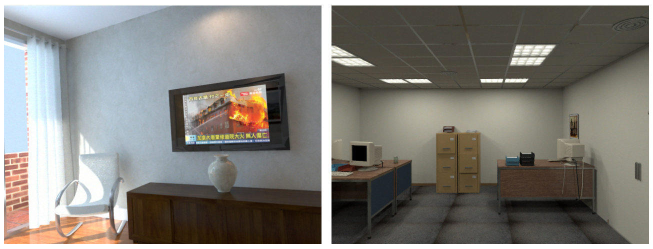

(a) Living Room (b) Office

Figure 6.6: Sample images of the 2 synthetic datasets offered by Davison et al. [15]

ground truths are not necessary, the original datasets associated with LSD-SLAM will be good enough. For *qualitative* analysis, any dataset that completes consistently with a loop closure may be used, and we will therefore use the *office room dataset* by Davison et al.

6.1.3 Testing Memory Consumption

Memory consumption plays a large role in determining the hardware requirements of an autonomously navigating system. Long access times to a stored map can create a bottle neck in evaluating possible traversable paths to goal points. Long access times can be attributed to the resolution and size of the map as it means many more cells in the map need to be traversed. Considering LSD-SLAM was built for large scale environments, this test will be done on a large scale dataset. The aim is to determine how much memory is consumed for a growing map during exploration of an environment, and also once exploration has ended and the map is finalised. Of course, these tests have already been performed by Hornung et al. [10] and well documented with regards to on-board *memory consumption* and *storage* of an occupancy grid map file. However, their sensor of choice was a dense lidar sensor as opposed to the semi-dense monocular vision sensor used here. We also aim to test how resolution of the map effects the overall memory consumption, as well as the map storage method.

The system developed in this thesis is capable of building an occupancy grid map in real time by writing the depth map of each completed key frame to a local map file. As mentioned in Section 3.5, the local map does not take loop closure into account and therefore is subject to scale and translational drift. However, once a loop closure occurs and the map is optimised to correct for drift errors, the new optimised map is saved (as an entire set of updated key frames) to a *global map file*. The global map file will be rebuilt from scratch upon every loop closure detection (that only occurs once within a minimum of 10 key frames) to ensure that the most accurate map is always available.

The map file grows continuously as the environment is explored. This poses a possible problem, when considering a large environment and the fact that the map file is stored in RAM as it is being built, seeing that the size of the map is then determined by the resolution and the size of the environment. If the new key frame was written directly to a file, the map would need to be reopened and stored in RAM to add a new key frame before saving it to the storage file again. This would create a bottleneck in access times and hinder the real-time performance of the system. If a low resolution was used and consequently reduced the number of cells that needed to be written to file, then writing every frame directly to file may be a trivial task; however, this method becomes cumbersome at higher resolutions with more dense measurements. From LSD-SLAM's side, parameters can be changed to adjust the number of pixels used for mapping and therefore the number of depth points in a key frame (reduce the density), which can reduce the amount of data processing associated with each key frame. A disadvantage of this is that it also reduces the the amount of information gained from LSD-SLAM. One can also change the frequency of key frame promotion by reducing the number of key frames being added to the map (or increasing the distance threshold between key frame promotion). *OctoMap* is also able to store occupancy grid maps in two file formats (binary states or stored probability values) to help combat memory consumption, as described in Section 4.4.4. The binary format of the map converts all probabilities stored in the map into a maximum likelihood of either occupied or free space (for example, *1* or *0* respectively). This conversion does result in a loss of probabilistic occupancy information, although the reduction in memory consumption is a significant benefit.

To perform tests on memory consumption, LSD-SLAM and our mapping system will be run on 2 datasets with a comparison of both sensor models. The machine dataset by TUM (being a large scale environment), and the synthetic office scene dataset by Davison (a small scale environment). This will allow us to evaluate results for maps that are of large or small environments and of high or low resolution. The machine dataset contains loop closure, while the office scene dataset does not. However, the office scene dataset maintains an acceptably constant scale throughout. This is an important consideration because if the drift tends towards a shrinking scale, for example, the resulting map would become less and less populated as the depth estimates reaches the maximum resolution of the map. The result would be a map file that never gets bigger, but also never gains new useful information. If the drift tends towards an increasing scale, the map size could potentially grow exponentially with each new frame and consume much more memory than is necessary. LSD-SLAM does protect against small scales by shutting down if the scale shrinks to a minimum value. Fortunately due to LSD-SLAM's *scale aware* image alignment algorithm, the probability of exponential growth in scale drift is very low. For this experiment the speed or accuracy will not be considered as we are only interested in the real time memory consumption with respect to the on-board RAM consumption for both sensor models. A comparison of RAM usage will also be made for both occupancy grid formats, maximum likelihood states, and full probability states. We will also keep LSD-SLAM parameters (specifically the *pixel gradient* threshold and *key frame promotion distance* threshold) at default values. While RAM usage is the focus here, the resulting file sizes will also be presented along with relevant data for each file, such as number of nodes, and the represented physical environment size of the map.

In this section of the evaluation, we will only be testing 1 parameter for each sensor model. The difference in map size for different parameter values is expected to be negligible in that the same amount of cells along a measurement beam get updated using the same calculation regardless of the parameter value. We will also only consider the *global map* that is created, as the local map can be chosen to represent only the immediate vicinity and therefore may be limited to any range around the current pose by the user. The global map would be the best representation of the complete environment and will always be the largest of the two maps. We will also limit the allowable memory consumption of a map to 10 Gb[1] of RAM usage to identify feasible resolutions or map sizes for general purpose requirements.

6.1.4 Testing Performance

LSD-SLAM is designed to run in real time, and it is an important goal maintain throughout autonomous navigation systems. In this section we will test the performance, or run time, of our system for various datasets. It should be noted that the system developed in this thesis is in no way optimised. Any results shown can undoubtedly be improved, although resource management on this level is out of the scope of this thesis. The results for these tests can be seen as a baseline performance comparison between two sensor models and dataset sizes, as we expect the *relative* performance of our system to be similar to an optimised system. The variables that will be compared in these tests are 2 datasets, 5 resolutions and 3 sensor models (Thrun, Andert and an ideal sensor model) with 1 set of parameters for each ISM. The two datasets for this test will demonstrate large and small scale environment performances, and so will be the same datasets used in the memory test (machine dataset and office scene dataset) in Section 6.1.3. The performance in each test will be recorded as an average time of the insertion of $100k$ measurement points into an occupancy grid map, as well as the time taken to complete the entire dataset. Each test for this section of the experiment will also record the performance of LSD-SLAM running in parallel to our system as it allows us to identify a bottleneck in the system.

[1]A limit of 10 Gb of RAM is chosen because anything more would not be feasible for most systems. In fact many entry level computers/laptops, and some higher end mobile phones, have between 2 - 8 Gb of RAM installed

6.1.5 Map Evaluation: Qualitative

As discussed in Section 6.1.2, the selection of viable datasets on offer for LSD-SLAM are few. Only one dataset (the synthetic living room dataset by Davison) can be used for *quantitative* evaluation, due to it being the only applicable dataset with an associated ground truth. There are many viable options for purely *qualitative* evaluation though, and some of the resulting maps will be displayed in these tests. Doing so may help gain a more intuitive perspective of how LSD-SLAM performs, as well as the sensor models incorporated. Tests will be run on both ISMs at a resolution of 16 mm, at 3 different parameter values for each ISM. We would like to determine, visually, how the parameters impact the resulting map.

6.1.6 Map Evaluation: Quantitative

Quantitative evaluation is arguably the most important test for this system. If an autonomously navigating system is to plan paths and avoid obstacles within an unknown environment, the most accurate possible map of the environment is needed, and this is what our quantitative tests aim to measure. While this section only uses one dataset (as discussed in Section 6.1.2), the results are still significant in that we are not only able to determine the accuracy of LSD-SLAM, but the impact of two sensor models with varying parameters.

To evaluate the accuracy of a map, a comparison with a ground truth map on a cell by cell basis must be made. This is why an accurate (if not exact) ground truth map is so important. An occupancy grid map, constructed by some algorithm, can be considered a classifier in terms of determining which cells within the map fall into a class of being free or occupied. Assuming a ground truth occupancy grid map and an occupancy grid map (constructed by some mapping system) exists, there are many methods to assess performance of a classification algorithm [78; 79]. Three main methods discussed here are based on what is most commonly used in literature for this field of study, namely the *sum of squared differences* evaluation, the *Matthews correlation coefficient* (MCC) and the *receiver operator characteristic* (ROC) curve. These methods will be described and the limitations of each evaluation method will be identified to determine what information can be obtained from each. It should be noted that quantitative evaluation of occupancy grid maps has mostly been performed on a 2D basis, with the exception of Marzat et al. [80], although their method is not made clear. Generally, for 2D map evaluation, the ground truth and the constructed maps are converted into images and compared using a pixel to pixel approach [79; 81; 82; 27; 83]. Because we are not limiting our system to an autonomous platform that only operates in a 2D state, we aim to evaluate the map on a 3D basis. Literature seems to be lacking in information on how this is done, although it can be assumed that it is not very different to the 2D case. In our approach, the 3D map is simply divided by the maximum height of the measured environment to create many 2D slices which are evaluated as a 2D map using a pixel by pixel comparison, and recorded over the height of the dataset. The value of the height is represented by units that correspond to the resolution of the map. If the occupancy grid map has a resolution of 5 cm cells and the maximum height of the environment in the map is 5 m, then the number of image slices would be 100.

6.1.6.1 Sum of Squared Differences

Given a ground truth map and a generated map, which are both aligned and of equal scale, a pixel by pixel comparison is made for each image slice where the difference in probability value as represented by the pixel colour is squared and added together, shown by Equation 6.1.1 as

$$SSD_m = \sum_{i=1}^{n} (I_{1p_i} - I_{2p_i})^2, \tag{6.1.1}$$

where

I_{1p_i} is the image slice of the constructed map,

I_{2p_i} is the image slice of the ground truth map,

p_i is pixel i,

n is the number of pixels within an image.

This results in a single value representing the similarity between corresponding image slices of the ground truth and the generated map. The resulting values are then averaged over each image for the height of the map to give a final metric of similarity for the generated 3D map, as shown by Equation 6.1.2 as

$$SSD_{avg} = \sum_{m=1}^{N} \frac{SSD_m}{N}, \tag{6.1.2}$$

where

SSD_m is the SSD for image slice m,

N is the number of image slices in the 3D map.

The lower the final metric, the more similar the maps are to each other [79].

Two maps can also be compared by normalising the resulting score, by dividing the total SSD by the total number of pixels that are evaluated over the entire map. However, we only have one map to evaluate in this test and it is therefore unnecessary to cater for. Another disadvantage of using this method is that it uses all cells within the map (free, occupied or unknown). Many mapping algorithms favour identification of free space rather than occupied space. This creates a bias in measurement when there is more free space than occupied space within the generated map. To circumvent this problem, we can set a threshold for occupancy where a cell is considered occupied when the probability value of the cell is above 0.5, for example, although the threshold itself is difficult to determine without a separate method of evaluation, like the one used in the ROC curve. This is also not feasible for our purposes as we would then be removing the probabilistic values of occupancy, SSD is therefore not ideal for our evaluation needs, and will not be used.

6.1.6.2 Matthews Correlation Coefficient

The Mathews correlation coefficient (MCC) is a method of assessing the performance of a binary classification algorithm, introduced by Matthews et al. [84]. It results in a value ranging between -1 and 1, where 1 shows a perfect classifier, 0 shows a completely random classifier and -1 shows a complete disagreement between the classifier and the observation.

MCC uses a confusion matrix or contingency table, as shown in Table 6.1, where a tally is performed on predictions made by the classifying model versus a ground truth comparison. Binary classes are measured against being a *true* or *false*, *positive* or *negative*. The resulting MCC metric is calculated using Equation 6.1.3 as

$$MCC = \frac{TP \times TP - FP \times FN}{\sqrt{(TP+FP) \times (FN+TN) \times (FP+TN) \times (TP+FN)}}. \tag{6.1.3}$$

Based on the measures defined in Table 6.1, we can calculate other performance characteristics such as *sensitivity* or *true positive rate* (TPR) and *specificity* or *true negative rate* (TNR). Sensitivity measures the proportion of predictions that correctly classified a positive class (or in our case correctly predicts an obstacle), while specificity measures the proportion of predictions that correctly classified a negative class (or in our case free space). Other measures can be made using the confusion matrix, such as the precision, recall and accuracy of the model, as well as the *F-score* which indicates the harmonic mean of precision and recall, shown in Equation 6.1.4 as,

$$F = 2 \times \left(\frac{PPV \times \text{sensitivity}}{PPV + \text{sensitivity}}\right), \tag{6.1.4}$$

where PPV is the positive predictive value defined as

$$PPV = \frac{TP}{TP+FP}. \tag{6.1.5}$$

Table 6.1: Confusion Matrix for MCC performance assessment on a binary classifying algorithm

	Predicted True	**Predicted False**	**Measures**	**Totals**
Actually True	True Positive (TP)	False Negative (FN)	True Positive Rate (TPR) Sensitivity TP/(TP+FN)	Total Actually True (TP + FN)
Actually False	False Positive (FP)	True Negative (TN)	False Positive Rate (FPR) Specificity $\frac{TN}{(TN+FP)}$	Total Actually False (FP + TN)
Measures	Positive Prediction Value Precision $\frac{TP}{(TP+FP)}$	Negative Prediction Value $\frac{TP}{(TP+FP)}$	Accuracy $\frac{(TP+TN)}{(TP+FN+FP+TN)}$	
Totals	Total Predicted True (TP + FP)	Total Predicted False (FN + TN)		

However, the *F-score* does not consider *true negatives* and is therefore often considered an inferior method of assessment of binary classifications models when compared to general MCC scoring, especially if safety is a priority [85]. This is because MCC favours sensitivity over specificity, which may be favourable to a system that cares more about obstacles or where *not* to go, than free space. If an incorrect prediction of an obstacle is made when there is actually free space, it poses no threat of collision to an autonomous system, whereas an incorrect prediction on free space when there is an obstacle does pose a threat of collision.

A limitation of using MCC as a performance assessment method is that it assumes a fixed threshold for binary classification. This means that for an occupancy grid map, the assessment will only be done for one occupancy threshold, which may be acceptable if dealing with a purely binary map on a single threshold. However, our system produces an occupancy grid map with cells containing probabilities. The occupancy of a cell depends on the threshold of occupancy and if the threshold varies, the result does too. The accuracy of the generated map is not only dependent on the sensor model applied to the measurement, but also the threshold applied to the probability of the cell to identify it as either free or occupied. Therefore using MCC for assessment is not well suited for our purposes, as we would like to determine the performance of our system on two sensor models over all thresholds of occupancy.

6.1.6.3 Receiver Operator Characteristic Curve

Similarly to MCC, the receiver operator characteristic curve (ROC curve) uses a contingency table, though Table 6.2 shows a simpler, more applicable version. ROC curves assess classification algorithms by comparing resulting predictions to a ground truth, over the whole range of thresholds from 0 to 1. For each threshold the comparisons are made and summed together for their respective class of being a *true positive (TP)*, *true negative (TN)*, *false positive (FP)* or *false negative (FN)*. The sensitivity is calculated with Equation 6.1.6 as

$$Sensitivity = \frac{TP}{TP + FN}, \tag{6.1.6}$$

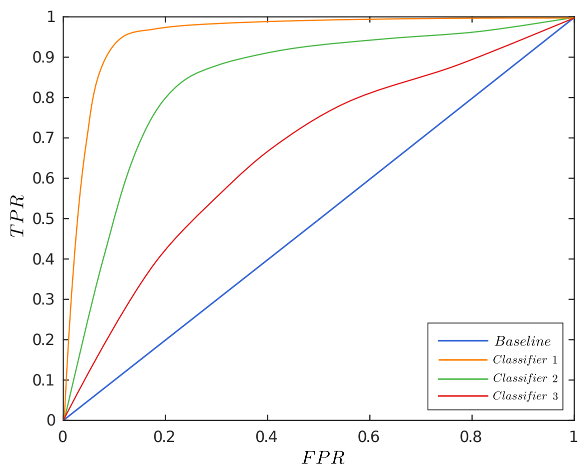


Figure 6.7: An example of a typical ROC curve, where 3 binary classifiers are evaluated. Classifier 1 (orange) shows the best performance as it is closer to the top left corner of the graph, while Classifier 3 (red) shows the worst performance as it is closer to the baseline. A classifier with a result equal to the baseline indicates that it performs no better than a random guess.

and the specificity is calculated with Equation 6.1.7 as

$$Specificity = \frac{TN}{TN + FP}. \tag{6.1.7}$$

ROC curves plot a visual representation of *sensitivity* versus $1 - specificity$ for the whole range of thresholds. The resulting graph displays a curve from $(0, 0)$ to $(1, 1)$ (see Figure 6.7 for an example). In an ideal case the resulting curve should cross $(0, 1)$ as it would indicate perfect prediction of true positives and zero false positives. The faster the curve rises to 1 at the beginning of the graph, and the higher the curve is, the better the model performs. Generally the graph is analysed by determining the area under the curve (AUC) since the ideal performance would result in an area of 1. The larger the area, the better the performance of the prediction algorithm [81; 86; 27]. It should be noted, however, that a ROC curve does not represent the actual performance of a prediction method, but rather it is *potential* performance or ranking potential. Although, ranking potential is related to performance and serves as a good indicator of the *goodness* of the prediction model, and its ranking in comparison to another model.

ROC curves prove to be the ideal performance assessment method for the purposes of this thesis as our maps consist of probabilities of occupancy where thresholds are used to determine the occupancy of a cell. Therefore the area under the curve of a ROC curve will be the metric of choice for measuring performance of our system when comparing the result of two sensor models to a ground truth map.

Table 6.2: Simplified Contingency Table for ROC curves

	Predicted True	**Predicted False**
Actually True	True Positive (TP)	False Negative (FN)
Actually False	False Positive (FP)	True Negative (TN)

To perform quantitative evaluation of the resulting map, the image slices of both the ground truth and the resulting map will be compared pixel by pixel for every threshold between 0 and 1 in increments of 0.04. A classification will be made using the contingency table (Table 6.2) and the resulting tally will be used to calculate the *sensitivity* (TPR) versus *1-specificity* (FPR) for representation. The evaluation will be done on one dataset; the *living room dataset*, for 5 resolutions, with both sensor models (by Thrun and Andert). Each sensor model, with the exception of the ideal model, will be applied with 10 different values in parameters, the L value and k value for Thrun and Andert respectively. A ROC curve is plotted for each test and the *area under the curve* (AUC) is calculated for each.

6.1.6.4 Dataset Preparation

The only dataset that caters to the requirements in this test is the *living room dataset* by Davison. It is the only dataset in our testing that LSD-SLAM consistently completes, and that has a ground truth map available, rather than just a ground truth trajectory. The ground truth map is only made available as a point cloud, and therefore is without explicit representation of occupied or free space. This has led to the need for processing the point cloud into an occupancy grid map. The ground truth map is also axis aligned, while the resulting map created by LSD-SLAM is on an arbitrary axis. The resulting map from LSD-SLAM also needs to be initialised with the correct depth map, although the resulting map still contains some scale drift. Therefore some processing needs to be done to align the resulting map with the ground truth map in both transformation and scale.

TUM offers a tool on their website to benchmark resulting RGB-D SLAM trajectories to ground truth trajectories [77]. The tool calculates the absolute trajectory error (ATE) as well as the relative pose error (RPE). In the evaluation of the ATE, the two trajectories are aligned using singular value decomposition and then the difference between each pose along each time-step is calculated. The output is the mean, median and standard deviation of these differences. This tool provides the basis to align two similar, but misaligned trajectories, which can easily be modified to output the translational and rotational differences between the two. With the translational and rotational differences between the two trajectories, it is a simple matter of applying the transform to the resulting map to align with the ground truth map.

Once aligned there is still the error in scale that needs to be dealt with. The same ATE calculation tool can be modified to find the scale difference by minimizing the trajectory error given a scale value. The minimization is done using the simple *golden section* minimization method. With this slight modification we are now able to take the resulting trajectory of LSD-SLAM and compute the transformation and scale difference to the ground truth, and then apply the correction to the result to align the trajectories, and therefore align the resulting map with the ground truth map.

Once the resulting and ground truth maps are at the same scale and alignment, the maps are cut into slices along the Z (height) axis at increments equal to the resolution of the map. Each slice is then converted into a gray scale image to represent the probability of occupancy, where black, dark gray and white represent free, unknown and occupied space respectively. The darker the gray the higher probability of being free space, while the lighter the gray the higher the probability of being occupied space. These images now allow us to compare map slices pixel by pixel to quantitatively evaluate the accuracy of the resulting map.

6.2 Part 2 - Results

In this section we will present the results of the tests outlined in Section 6.1. The experiments are performed to compare two ISMs (one by Thrun [28; 27] and one by Andert [76]) on three main criteria, *memory* in Section 6.2.1, *performance* in Section 6.2.2 and *accuracy*. The accuracy is split into two evaluations on *qualitative* and *quantitative* analyses in Section 6.2.3 and Section 6.2.4 respectively.

6.2.1 Memory Consumption Tests

Ideally, to allow this system to work on an autonomously navigating platform, the memory use need to be as low as possible, both in RAM usage and storage space. We test two datasets, a large and small environment respectively. The large *real* environment dataset (the *machine dataset*) is provided by TUM and was used by LSD-SLAM authors for testing as well. The small environment dataset is a synthetic office scene, by Davison, used for benchmarking purposes. Table 6.3 shows some relevant information about the datasets used.

For the large scale machine dataset, initial tests show that under default settings of LSD-SLAM, it was not feasible to run on high resolutions due to the high RAM consumption (over 10 Gb). By default, the number of key frames selected from a set of 7186 image frames resulted in approximately 220 key frames. With an average depth estimation density of approximately 35% (where a density of 100% indicates that every pixel in every image is used to estimate depth), the total number of points to process came to approximately 24.3 million. The minimum distance between key frame promotion was then extended which dropped the number of selected key frames for the

Table 6.3: Information on datasets used for memory tests. The number of key frames for the machine dataset are achieved by increasing the distance between key frame promotions, while the office dataset was run on default parameters.

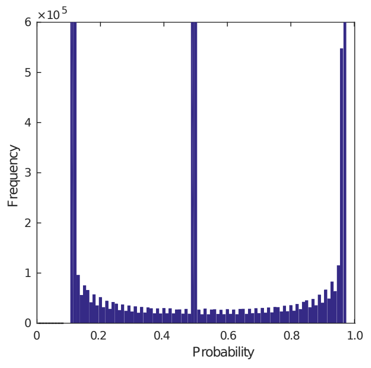
			LSD-SLAM	
Dataset	**Environment Size (m)**	**Dataset Run-time**	**No. of kf**	**No. of Points**
Machine	30 x 30 x 12	2:20 min @ 50Hz	≈ 148	≈ 16.35 million
Office Room	7 x 8 x 3.5	0:51 min @ 30Hz	≈ 45	≈ 3.44 million

Table 6.4: Tabulated results for memory tests for 2 ISM's on 2 datasets (machine and office). The results show the different memory usage for maximum likelihood and full probability maps in terms of RAM and file size for 5 different resolutions (Res). The 2 formats in which the maps are stored are shown as *.bt* for binary and *.ot* for full probability storage. The orange blocks indicate the maps with the lowest memory use. An 'x' indicates a map that exceeded 10 Gb of RAM consumption and had to be aborted.

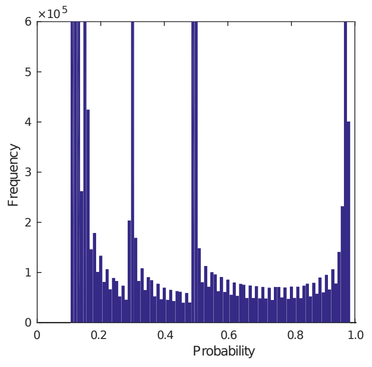
		Datasets							
		Machine				**Office**			
		RAM usage (MB)		**Files Size (MB)**		**RAM usage (MB)**		**File Size (MB)**	
Model	**Res (m)**	**.bt**	**.ot**	**.bt**	**.ot**	**.bt**	**.ot**	**.bt**	**.ot**
THRUN	**0.008**	x	x	x	x	1500	1700	3.6	50.9
	0.016	x	x	x	x	555.8	597.7	0.734	9.7
	0.032	4700	6100	30.0	465.1	125.8	133.5	0.153	1.9
	0.064	1000	1000	5.1	61.7	41.7	41.5	0.032	0.398
	0.128	674.1	679.1	0.985	12.0	20.1	24.6	0.007	0.084
ANDERT	**0.008**	x	x	x	x	1100	2300	3.6	153.1
	0.016	x	x	x	x	531.8	938.9	0.725	22.3
	0.032	5700	x	39.9	1300	123.3	181.1	0.151	3.9
	0.064	836.2	2400	5.0	194.3	40.2	49.3	0.032	0.746
	0.128	674.3	1100	0.989	27.0	23.9	25.4	0.007	0.149

dataset down to 148 (≈ 16.35 million points) while still maintaining acceptable and consistent performance. By reducing the number of selected key frames, the accuracy and robustness of the tracking component of LSD-SLAM is also reduced. However, even with the reduction in key frames, the memory consumption was still high for the higher resolutions on the machine dataset. For the office scene, the default parameters of LSD-SLAM were still used. The results of the memory tests are shown in Table 6.4. A comparison of both datasets for both sensor models is shown, where an 'x' depicts results where the RAM consumption was too high to record (over 10 Gb). Both file formats are also shown, where *.bt* and *.ot* are the maximum likelihood binary state and full probability state formats respectively.

As seen from Table 6.4, in terms of file size, Andert's model seems to be more memory intensive for full probability storage than Thrun's method, however there is little difference between the two models when using maximum likelihood binary storage for either of the datasets. One factor to consider is that Thrun's model makes predictions of free space with a minimum probability assignment of 0, while Andert's model uses a minimum assignment of 0.3. The other contributing factor is that the significance factor k in Andert's model has a larger effect on predictions of occupancy than it does on predictions of free space. This means that Thrun's model would cause more cells to reach the clamping threshold on free space sooner than Andert's model. When more of the cells are at the clamping threshold, the map can be more efficiently compressed since more child nodes of the same value can be merged into a parent node of the same value. Figure 6.8 shows how Andert's model creates a larger, or more even, distribution of probabilities over the map (particularly between probabilities of $0 - 0.5$ and $0.5 - 1$), while Thrun's method is more focused toward the clamping thresholds.



(a) Thrun



(b) Andert

Figure 6.8: Average distribution of occupancy probabilities, in the generated occupancy grid map, for Thrun (a) and Andert's (b) ISM, over all tested resolutions.

In both cases of file storage (binary and full probability states) the maps are pruned. Pruning a map attempts to identify a set of child nodes with equal probabilities, and replaces them with a parent node of the same probability, to save storage space. Andert's model, creating a larger distribution of probabilities, causes a lower probability that child nodes with the same probabilities exist, which is contrary to Thrun's model, and therefore results in a larger full probability state file size. The binary state file sizes are similar because the cells are converted to a maximum likelihood state, and therefore allows for the majority of the map to be successfully pruned. The effect of distributed probabilities carries over to the RAM usage as well, where the results are clearer on the larger machine dataset.

Table 6.5: Information on datasets used for performance tests to evaluate the actual execution time achieved using the open-source version of LSD-SLAM. The number of key frames for the machine dataset are what we achieved by increasing the distance between key frame promotions, while we tested the office dataset on default parameters.

			LSD-SLAM		
Dataset	Environment Size (m)	Dataset Run-time	No. of KF	No. of Points	Achieved Run-time
Machine	30 x 30 x 12	2:20 min @ 50Hz	≈ 148	≈ 16.35 million	11:22 min @ 10Hz
Office	7 x 8 x 3.5	0:51 min @ 30Hz	≈ 45	≈ 3.44 million	1:51 min @ 14Hz

Table 6.6: Performance test results to compare an ideal, Thrun and Andert's ISM on the machine dataset. The orange block indicates the best performing ISM between Thrun and Andert's ISM.

	Time per 100k point insertion (sec)				
Resolution (m)	**0.008**	**0.016**	**0.032**	**0.064**	**0.128**
Ideal	4.23	2.01	1.35	0.47	0.15
Thrun	14.02	5.76	3.04	2.00	1.14
Andert	25.28	8.17	7.26	4.15	1.68

Table 6.7: Performance test results to compare an Ideal, Thrun and Andert's ISM on the office dataset. The orange block indicates the best performing ISM between Thrun and Andert's ISM.

	Time per 100k point insertion (sec)				
Resolution (m)	**0.008**	**0.016**	**0.032**	**0.064**	**0.128**
Ideal	0.94	0.40	0.20	0.11	0.05
Thrun	4.72	2.17	1.31	0.55	0.29
Andert	7.37	3.03	1.45	0.74	0.38

6.2.2 Performance Tests

To test the performance of Andert and Thrun's sensor model within our system, an average time is calculated per $100,000$ inserted points for each model, and compared to an ideal inverse sensor model as well. Relevant information on the datasets used in this section of testing is provided in Table 6.5, where we also show our achieved performance of LSD-SLAM on these datasets. The results achieved, on the occupancy grid maps that are created, are presented in Table 6.6 and Table 6.7 for the *real* machine and *synthetic* office datasets respectively.

From Table 6.5, where we test the execution time of LSD-SLAM, we can see, at least the open-source version, that it does not perform at the level of what the authors claim to be *real-time*[2] (30 Hz) [87]. However, it can be argued that its performance is still "good enough" for a real-time system, provided it moves at a reasonably slow pace. It should also be noted that our system is not optimised, and results are clearly dependant on many factors such as the size of the dataset/environment, density of depth estimates, pixel/image noise and

[2]Even though 30 Hz is generally considered *real-time*, it still limits camera motion speeds and movements.

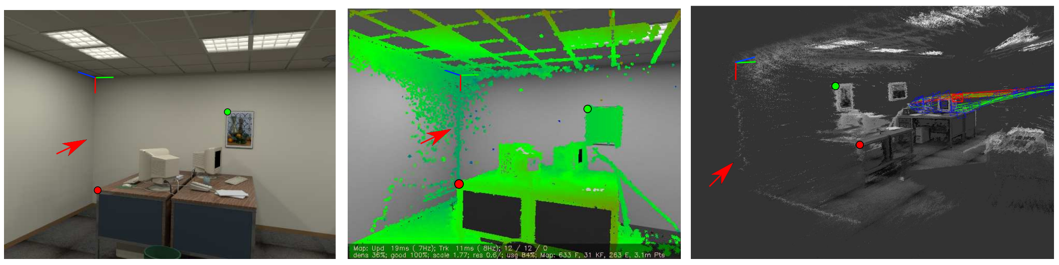
sensor model used. From Tables 6.6 and 6.7, where we test the performance of our mapping system for the 2 ISMs, it can be seen that Thrun's model performs significantly better than Andert's model in terms of insertion time. At lower resolutions, the speed is very similar, but the difference becomes clearer at higher resolutions. The speed difference can be attributed to not only the longer calculation time of Andert's model, but also the manner in which the resulting map is stored in memory, as discussed in Section 6.2.1, and therefore may slow down access times when updating cells.

6.2.3 Map Evaluation: Qualitative

In this section we will look at the quality of the resulting maps from a purely visual perspective. It may help highlight some features and information on differences that a quantitative evaluation cannot convey. The visualisations presented in this section use *Octovis*, which is a built in display tool for *OctoMap*. During the display of a map, the *occupied* cells shown, are only regarded as occupied by the visualisation tool if the probability value for the cell is higher than and occupancy threshold set by the program, such as 0.7 for example. It does not necessarily mean it is in a binary state of *completely occupied* or at a clamping threshold of occupancy. Octovis displays the colours of cells in a pre-programmed way. For Andert's ISM, red cells indicate cells that are considered occupied after one measurement with a probability of 0.97 (the upper clamping threshold), blue cells are considered occupied after multiple measurements with a probability of 0.97, gray cells are considered occupied with some probability simply higher than 0.7 and green cells indicate free space. For Thrun's ISM, light blue cells indicate occupied cells (at clamping threshold) based on a single measurement, while dark blue cells indicate occupied cells with any probability higher than 0.7.

Figure 6.9 shows the office scene dataset being used on LSD-SLAM, where depth estimates are made on pixels with sufficient intensity gradient changes, to generate a point cloud. Figures 6.10 and 6.11 show the resulting 3D occupancy grid map of both ISMs (Thrun and Andert respectively) for the *office dataset* across a range of parameter changes at a resolution of 16 mm. The point cloud in Figure 6.9(c) shows the corner of the office wall, which is the same perspective of the occupancy grid maps shown for both ISMs. For Thrun's ISM (see Figure 6.10) the L value is applied as 0.05, 0.45 and 0.9 respectively. For Andert's ISM (see Figure 6.11) the significance factor k is applied as 0.005, 0.05 and 0.09 respectively.

From Figures 6.10 and 6.11 we can see how varying the relevant parameters changes the outcome of the resulting map. Looking at the corner of the wall, we can see that Andert's model does not necessarily get much *thicker* with an increase in k value. We do however see that the results of occupancy become much more prominent, which is to be expected. Figure 6.9(b) also shows some gray cells, which are, as mentioned above, the cells that are regarded



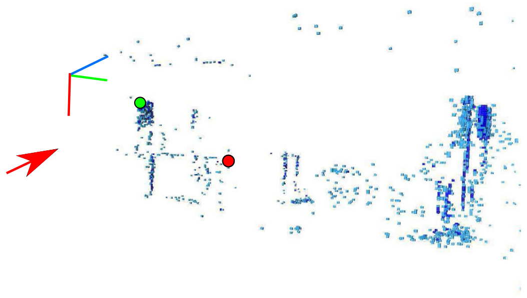
(a) Office scene dataset (b) LSD-SLAM tracking (c) LSD-SLAM pointcloud

Figure 6.9: Sample images of the office scene dataset, where (a) shows a sample image of the dataset with a red arrow indicating the corner of the wall. Figure (b) shows highlighted pixels that LSD-SLAM used to track the scene in the dataset. Figure (c) shows the resulting point cloud as generated by LSD-SLAM, also with a red arrow indicating the corner of the wall. The image of the point cloud serves as the point of view for both Figures 6.10 and 6.11.

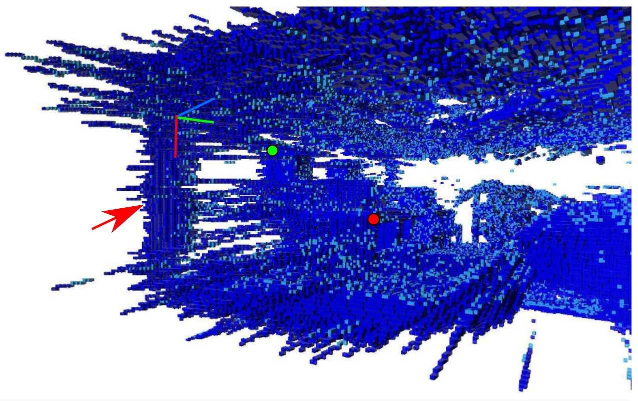
as *occupied* by definition of a predetermined threshold, but not quite at the clamping threshold. It can be seen that the gray cells progressively become replaced by blue cells to represent clamping thresholds of occupancy. Thrun's model performs differently in that the corner most certainly does get wider with an increase in L value. We can also see that for a low L value, the original intention of the buffer value does indeed have little effect on assignments of occupancy.

6.2.4 Map Evaluation: Quantitative

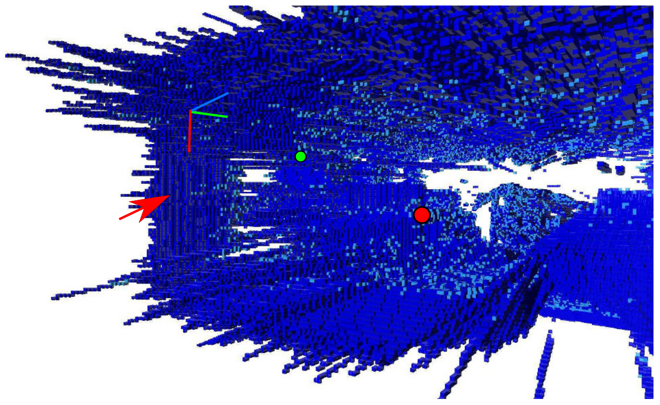
For *quantitative evaluation* of our system, and comparison of two ISMs, the living room dataset by Davison [51] is used as it is a synthetic dataset with perfect ground truth. The dataset is packaged as a sequence of image frames that are fed into our system which then creates an occupancy grid map based on probability assignments made by one of two ISMs (one by Thrun and one by Andert). We also compare the accuracy of these two sensor models to an ideal inverse sensor model. For consistency and fairness, the dataset was run on LSD-SLAM and the resulting inverse depth map was recorded before being applied to our occupancy grid mapping system. Within our occupancy grid mapping system, each sensor model has a parameter changed to 1 of 10 values, with the exception of the ideal sensor model which uses a *default* L value parameter as the diagonal length of the cell size within the map. For Thrun's model we altered the L value, and for Andert's model we altered the significance factor k. Each sensor model along with the changes in their parameters were run at 5 different resolutions, for a total of 50 resulting maps for each sensor model (Thrun and Andert) and 5 resulting maps for the ideal model. The *global map* file created by the algorithm is the one that is used, although in testing it was no different to the *local map* as no loop closure was



(a) Thrun - $L = 0.05$

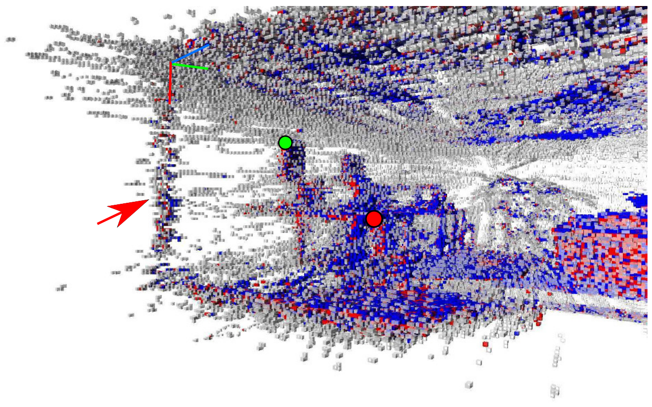


(b) Thrun - $L = 0.45$

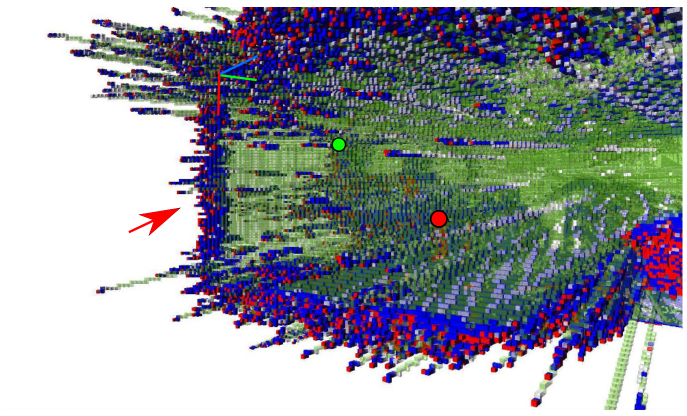


(c) Thrun - $L = 1.0$

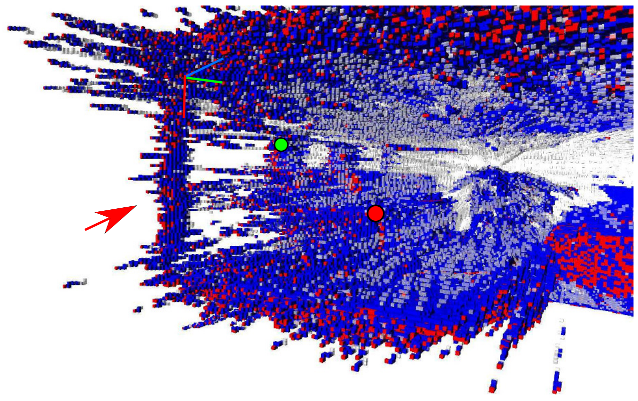
Figure 6.10: Occupancy grid maps generated using Thrun's ISM on the office scene dataset, using a buffer value of $L = 0.05$ in (a), $L = 0.45$ in (b) and $L = 1.0$ in (c). The red arrow indicates the location of the corner of the wall as shown in LSD-SLAM's point cloud in Figure 6.9(c). Light blue cells indicate occupied cells (at clamping threshold) based on a single measurement, while dark blue cells indicate occupied cells with any probability higher than 0.7, and free space is not explicitly shown here.



(a) Andert - $k = 0.005$



(b) Andert - $k = 0.05$



(c) Andert - $k = 0.09$

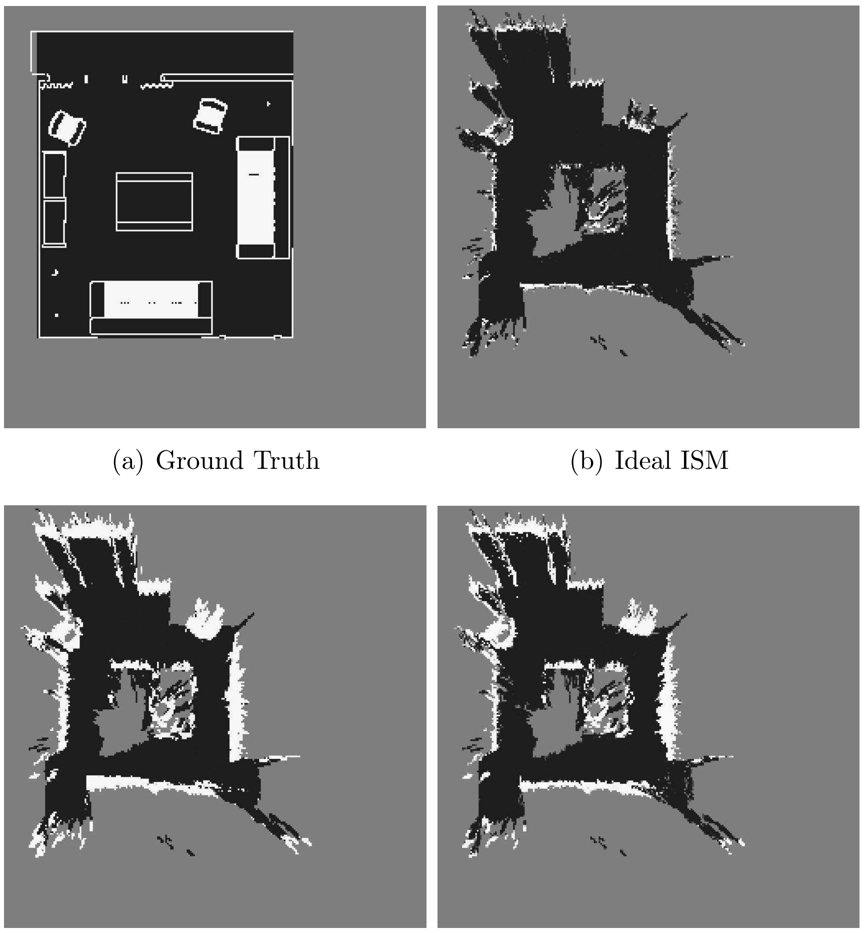
Figure 6.11: Occupancy grid maps generated using Andert's ISM on the office scene dataset, using a significance factor of $k = 0.005$ in (a), $k = 0.05$ in (b) and $k = 0.09$ in (c). The red arrow indicates the location of the corner of the wall as shown in LSD-SLAM's point cloud in Figure 6.9(c). Red cells indicate cells that are considered occupied after one measurement with a probability of 0.97 (clamping threshold), blue cells are considered occupied after multiple measurements with a probability of 0.97, gray cells are considered occupied with some probability higher than 0.7 and Figure (b) shows free space as indicated by the green cells.

Table 6.8: Resulting AUCs from ROC curve analysis for Thrun's ISM for each L value at 5 different resolutions. The orange highlighted values are the highest AUC values per column.

THRUN	**AUC for L values at Resolutions (m)**				
L Value	**0.008**	**0.016**	**0.032**	**0.064**	**0.128**
0.05	0.63	0.61	0.66	0.69	0.60
0.1	0.70	0.67	0.71	0.74	0.66
0.2	0.77	0.73	0.77	0.79	0.74
0.3	0.85	0.75	0.81	0.82	0.78
0.4	0.90	0.78	0.83	0.86	0.79
0.5	0.91	0.77	0.82	0.87	0.80
0.6	0.90	0.74	0.80	0.85	0.79
0.7	0.89	0.66	0.78	0.83	0.78
0.8	0.88	0.60	0.73	0.80	0.78
1.0	0.85	0.56	0.61	0.72	0.76

explicitly detected. The resulting maps are all scaled and aligned using an optimisation technique (Section 6.1.6.4) to match the ground truth map, and then divided into image slices. The image slices are made by converting the probability of occupancy into a gray-scale value for each grid cell. A ROC curve analysis, with the calculated AUC, is done on each map using their respective set of image slices. An example of the image slices for each sensor model and the ground truth map, which are all occupancy grid maps, are shown in Figure 6.12 for comparison. The ground truth map was created by converting the 3D model of the dataset into an occupancy grid map, and filling all empty space within the room with free space. Figure 6.13 and Figure 6.14 show the resulting ROC curves of all the tests done on Thrun and Andert's ISMs respectively, and Table 6.8 and 6.9 shows a list of the area under the curve (AUC) results achieved for each version of the map with Thrun and Andert's models respectively.

In Figure 6.13 we see that as the L value (or buffer value to accommodate for cell size) is increased, the resulting sensitivity of Thrun's ISM also increases, that is to say the rate at which true positive predictions are made (TPR), increases. The increase in L value also causes the sensor model to tend toward a higher false positive rate (FPR), or $1 - specificity$. The effect is due to the fact that for higher L values a cell that is considered occupied is overestimated to have a probability of occupancy over many cells along the measurement beam, centred around the range measurement. Intuitively this would cause a few of the cells within a measurement beam to be correct, while many are incorrect, and therefore a trade-off is made. The more correct positive predictions we wish to make, the more incorrect positive predictions we will need to deal with. This pattern follows for Thrun's model across all resolutions tested. The resulting AUC's (area under the curve) for all tests done on Thrun's model, as seen in Table 6.8, shows very favourable performance with the highest AUC of 0.91 at the highest tested resolution of 8 mm for an L value of 0.5. From the table we can also see that across all resolutions, an L value of between 0.4 and 0.5 seems to yield the best results.



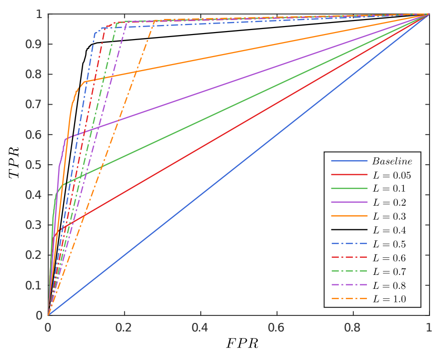
(a) Ground Truth (b) Ideal ISM

(c) Thrun ISM (d) Andert ISM

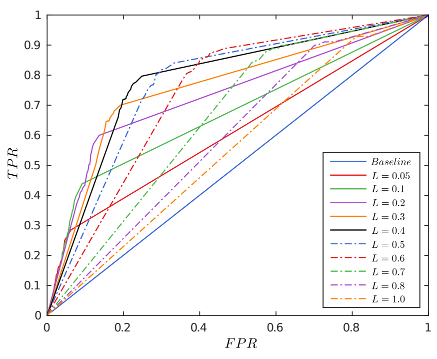
Figure 6.12: Sample image slices of the *living room dataset* with the ground truth map, ideal inverse sensor model, the map created with Thrun's model and Andert's model from left to right respectively. Dark gray to black represents free space, white represents occupied space and gray is unknown space.

For Andert's model in Figure 6.14, we see a slightly different pattern. At the highest resolution, 8 mm, the ROC curve for Andert's model shows two distinct features. The first is the *step shape* of the curve made by each tested k value, and the second is how the curves keep the same shape and height in peak TPR for increasing k values (up to a point).

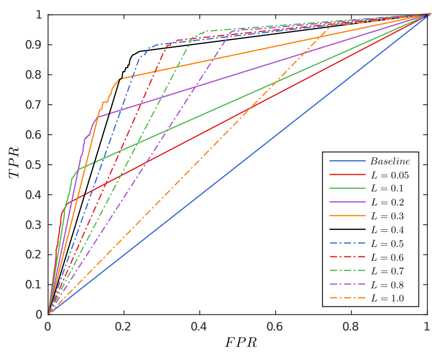
The shape of the ROC curve for Andert's model can be attributed to the distribution of probabilities at such a high resolution. Figure 6.15 shows the difference in frequency of probability distributions for Thrun and Andert's model. Thrun's model peaks in 3 locations, the minimum and maximum clamping thresholds (0.12 and 0.97), and for unknown cells (0.5), while Andert's model peaks in 4 locations; equal to Thrun's model with an additional peak at 0.3 as the minimum free space assignment probability. Because the



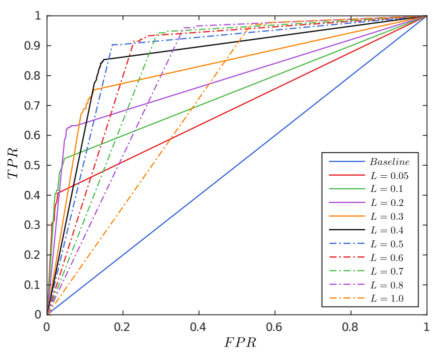
(a) 0.008 m



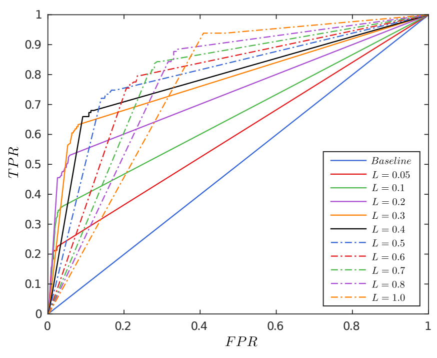
(b) 0.016 m



(c) 0.032 m

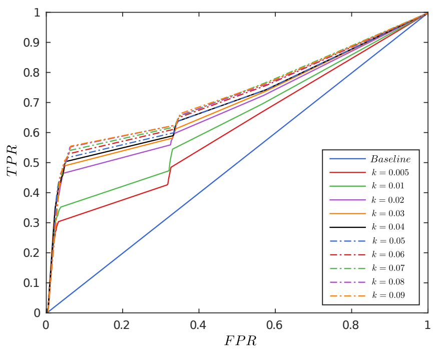


(d) 0.064 m

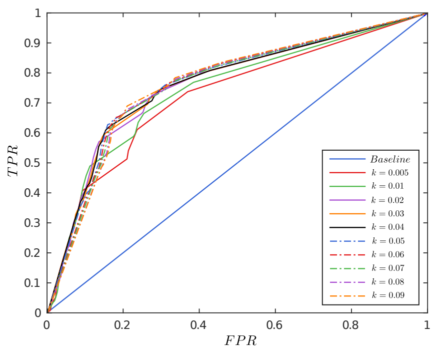


(e) 0.128 m

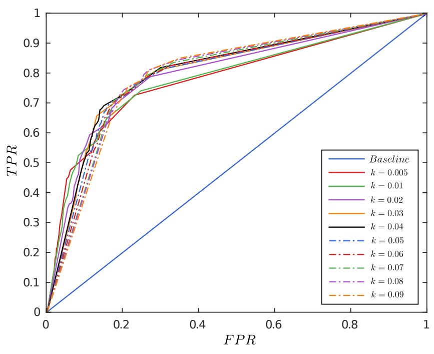
Figure 6.13: ROC curves for Thrun's ISM on a synthetic dataset with varying L values, at 5 different resolutions, where the tested resolutions of the maps are 0.008 m (a), 0.016 m (b), 0.032 m (c), 0.064 m (d) and 0.128 m (e). The baseline indicates a threshold of a random classifier, and any curve below the baseline indicates a negatively correlating classifier.



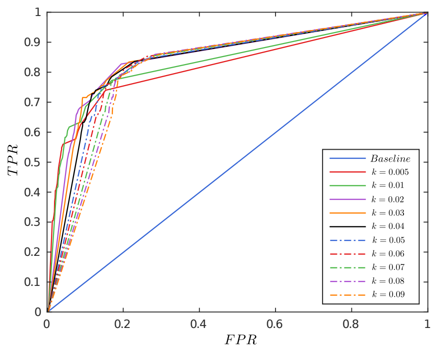
(a) 0.008 m



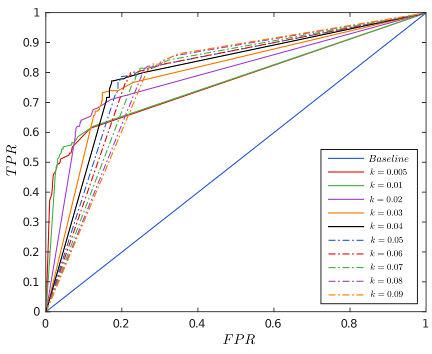
(b) 0.016 m



(c) 0.032 m

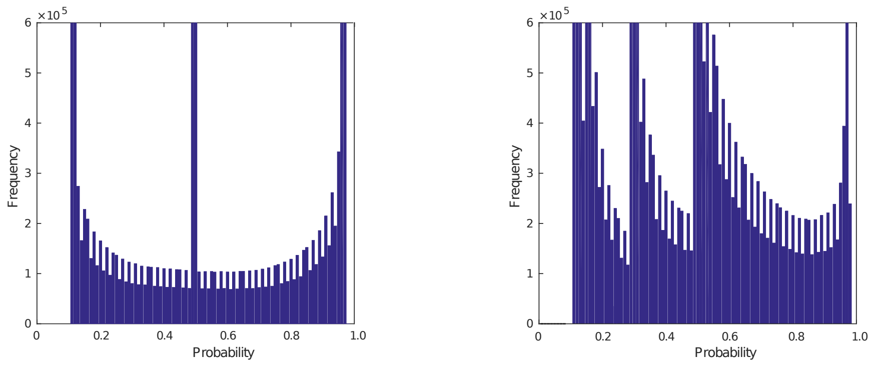


(d) 0.064 m



(e) 0.128 m

Figure 6.14: ROC curves for Andert's ISM on a synthetic dataset with varying k values, at 5 different resolutions, where the tested resolutions of the maps are 0.008 m (a), 0.016 m (b), 0.032 m (c), 0.064 m (d) and 0.128 m (e). The baseline indicates a threshold of a random classifier, and any curve below the baseline indicates a negatively correlating classifier.



(a) Thrun's ISM (b) Andert's ISM

Figure 6.15: Frequency of probabilities distributed within a map created using Thrun's ISM (a) and Andert's ISM (b) at a resolution of $8mm$. Thrun's model shows a peak at both clamping thresholds (0.12 and 0.97) and at unknown space, while Andert's model shows an extra peak at 0.3 from the minimum free space assignment value.

resolution of the map is so high, the probability of a cell being updated multiple times is much lower than that of the lower resolutions. Therefore there are many more free cells with a free space assignment of 0.3.

The second identifiable feature which is that Andert's ROC curves, at the highest resolution, keeps the same shape and peak in TPR for increasing k values, is a clear indication of how the significance factor effects the model. The significance factor k acts as a scaling factor for measurements, and because the occupancy grid map has a clamping threshold, the significance of a probability estimate can only be scaled up to a limit (which is the clamping threshold). At the higher k values, the map becomes saturated and the results do not get better, as can be seen in Table 6.9. The ROC curves for the remainder of the resolutions show similar characteristic improvements to Thrun's model because the resolution is lower, therefore more measurements have an impact on the occupancy probability of a cell, and in turn reach the clamping threshold at a slower rate. Table 6.9 shows that Andert's ISM achieves its best AUC of 0.85 at a resolution of 64 mm with a significance factor $k = 0.025$, although the AUC's for the tested k values still achieve similarly good results for resolutions higher than 8 mm. Based on these results, the *best* k value seems to be rather difficult to choose as Table 6.9 shows that the effect of the significance factor is resolution dependent. In other words, the significance factor chosen for Andert's model depends on the how many measurements one would prefer to cause a cell to reach a clamping threshold of occupancy.

As seen in Table 6.8 and 6.9, Thrun's model results in the highest AUC for all 5 resolutions when compared to Andert's model, with Thrun's highest AUC being 0.91 when an L value of 0.5 is used at a resolution of 8 mm. Andert's highest AUC equated to 0.85 for a significance factor k of between 0.02 and

Table 6.9: Resulting AUCs from ROC curve analysis for Andert's ISM for each k value at 5 different resolutions. The orange highlighted values are the highest AUC values per column.

ANDERT	**AUC for k values at Resolutions (m)**				
k Value	**0.008**	**0.016**	**0.032**	**0.064**	**0.128**
0.005	0.62	0.72	0.78	0.82	0.77
0.01	0.65	0.74	0.79	0.84	0.77
0.02	0.69	0.76	0.80	0.85	0.79
0.03	0.70	0.76	0.80	0.85	0.80
0.04	0.71	0.76	0.81	0.84	0.80
0.05	0.71	0.77	0.80	0.83	0.80
0.06	0.72	0.77	0.80	0.83	0.79
0.07	0.73	0.76	0.80	0.82	0.79
0.08	0.73	0.76	0.80	0.82	0.79
0.09	0.73	0.76	0.80	0.81	0.78

0.03 at a resolution of 64 mm.

Tables 6.8 and 6.9 also show that a high AUC does not necessarily describe the best model in a ROC analysis. On either side of the highest AUC, some parameters result in a similar AUC, such as with Thrun's model (see Table 6.8) for a resolution of 8 mm where an L value of 0.3 and 1.0 yield the same AUC. If one had to select the best of these two models, the AUC would not be useful. As seen in Figure 6.16, these two parameters clearly cause a different result. The model with $L = 0.3$ has a curve that travels from $[0, 0]$ towards the upper left corner $[1, 0]$ faster than the model with $L = 1.0$ that has a curve, that although reaches a higher TPR, also tends toward FPR faster than the former model. In this case using an $L = 0.03$ will result in fewer true positive predictions, but fewer false alarms. The position of the optimal point of the curve must be considered to evaluate the trade off. In terms of the resulting occupancy grid map, using $L = 0.3$ at the optimal threshold, would create a map with fewer false obstacles but more incorrectly labelled free space, while using $L = 1.0$ would create a map with fewer incorrectly predicted free space but more false obstacles. While the latter may seem like the safer option, it comes with the risk of filling an environment with false obstacles which could severely limit the possible paths the robot could use to navigate the environment.

The relationship between FNR and TPR is defined by Equation 6.2.1 as

$$FNR = 1 - TPR, \tag{6.2.1}$$

where TPR is the *sensitivity*,

and the relationship between TNR and FPR is defined by Equation 6.2.2 as

$$TNR = 1 - FPR, \tag{6.2.2}$$

where TNR is the *specificity*.

Recall true positives indicate a correct prediction of occupancy, while a false negative is an incorrect prediction of free space. In terms of autonomous

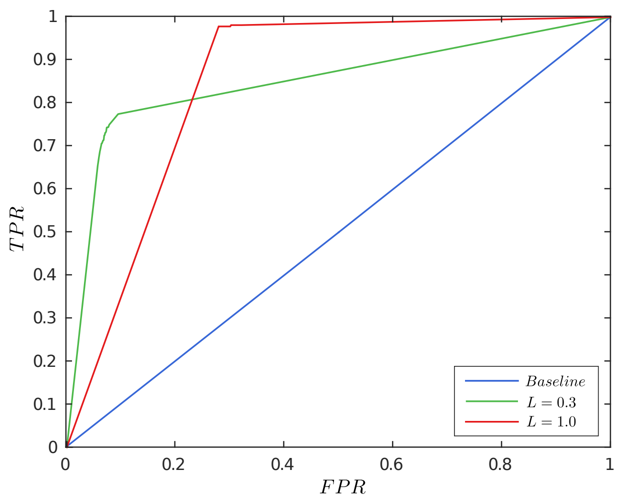
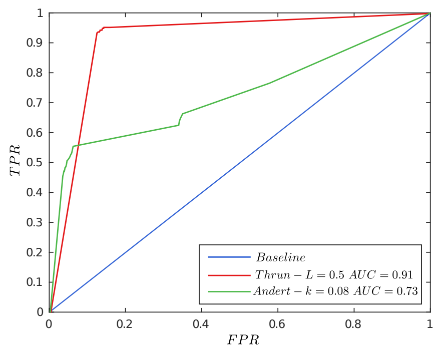


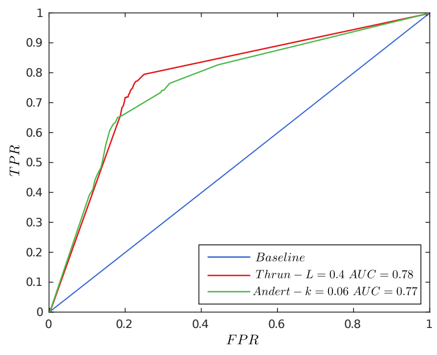
Figure 6.16: ROC curve of Thrun's ISM with two different L values (0.3 in green and 1.0 in red) resulting in the same AUC of 0.85. The baseline indicates the threshold were a classifier would perform no better than a random guess.

navigation, a false negative is arguably the result with the highest risk. This specification helps us decide which model would perform best between the two models in Figure 6.16. The model with the highest TPR results in the lower FNR and therefore may be the most desirable model. Choosing this model of course comes with the trade-off of having more false positives, and a false positive is the better option as it would be acceptable to navigate around free space than to navigate through an obstacle. However, having too many false positives significantly reduces the number, and quality, of possible paths an autonomous system could use to navigate. This decision is also only valid up to a point, where eventually a curve that goes too far towards FPR or $[1, 1]$ would also result in a map that is no different to a random guess, hence a balance must be found.

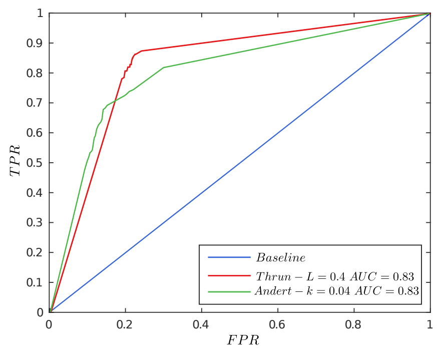
With the above information the best curve is found by using the highest AUC for each model as an indication of the probability of performance ranking when compared to another model. When two models with the same AUC are compared, the one with the highest TPR is chosen. We employ this criteria to compare the *best* model for Andert and Thrun for each resolution against each other. Figure 6.17 shows this comparison. For the higher resolution tested (0.008 m, 0.016 m, 0.032 m and 0.064 m), Thrun's ISM always performed better than Andert's ISM to this criteria. Andert's model performed slightly better for a resolution of 0.128 m based on the criteria of having a higher TPR, while the resulting AUC is the same as Thrun's model. Interestingly, this is in contrast to results found by Burger [36] where the conclusion was drawn that Thrun's model favours false negatives, and Andert's model favours false positives. However, Burger uses an L value of 0.25 for Thrun's model, and a significance factor of $k = 0.1$ is used for Andert's model throughout for either a resolution of 2.5 cm or 25 cm on two datasets. The resulting ROC curves in Figure 6.13 and 6.14 show that it is possible to achieve better results with a different parameter value. However, Burger's results were based on dense stereo sensors, which incorporates depth dependent uncertainty. This creates a different characteristic for each of the sensor models tested (as discussed in Chapter 5) where both Thrun and Andert's models drop in probability assignment over distance. However, as shown in Chapter 5, Thrun's model assigns a lower probability ($P < 0.5$) at the actual measurement than what it assigns slightly beyond the measurement, while Andert's model always ensures a probability of atleast $P = 0.5$ at the actual measurement. This effect justifies Burger's results, as well as the results achieved here, as we do not incorporate depth dependent uncertainty.



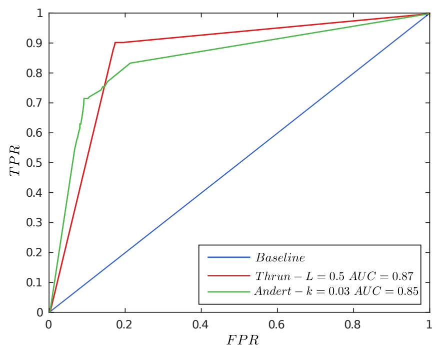
(a) 0.008 m



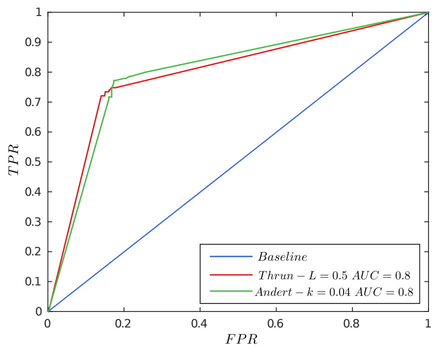
(b) 0.016 m



(c) 0.032 m



(d) 0.064 m



(e) 0.128 m

Figure 6.17: Comparison of best ROC curves for Thrun and Andert's ISM based on the highest AUC for varying parameters of L and k respectively, at 5 different resolutions, where the tested resolutions of the maps are 0.008 m (a), 0.016 m (b), 0.032 m (c), 0.064 m (d) and 0.128 m (e). The baseline indicates a threshold of a random classifier, and any curve below the baseline indicates a negatively correlating classifier.

Chapter 7

Conclusions and Future work

7.1 Conclusions

The purpose of this thesis was to investigate the use of LSD-SLAM and monocular vision to create a 3D occupancy grid map from semi-dense depth data of a static environment. We also aimed to compare the performance of two inverse sensor models based on 3 main criteria, namely *memory consumption*, *performance* and *accuracy*, where accuracy is divided into a *qualitative* and *quantitative* evaluation. The resulting map was then evaluated on these criteria to determine its usefulness for autonomous navigation in real-time. This chapter summarises our investigation with some concluding points before ending with a word on potential future work on the subject.

Our research began with a literature review where an overview of general SLAM was given, with emphasis on Monocular SLAM techniques, and an introduction to LSD-SLAM (by TUM) as the technique of choice for our research. These choices were based on the premise that monocular vision has significant advantages over other sensor types, especially when trying to ramp up the speed of progress within the field of autonomous navigation by making systems simpler and cheaper to employ. LSD-SLAM was an ideal candidate for our experiments as it was a novel method with promising results that was well suited for our purpose. Being a monocular vision based method meant scale ambiguity was an inherent problem; however, it is justified by the fact that it allows for efficient transition between differently sized environments and there are numerous methods available to incorporate scale. We also identified a few prominent features of LSD-SLAM that made it stand out among other SLAM techniques. LSD-SLAM is a *direct method*, in that it does not use features within an image, but rather pixel gradient information. It also incorporates stereo comparisons using variable baseline techniques, which helps mitigate the quadratic error over distance that is generally faced by stereo vision sensors. We then presented and discussed, in detail, the inner workings of LSD-SLAM along with some possible problems and restrictions that are expected to be experienced when using the algorithm. We also highlight another benefit, which proved to be in line with our goals, that an open source version of LSD-SLAM was made available by the authors. This allowed for modifications to be made to the algorithm, where the integrated loop closure (using the OpenFABMAP library) enables us to achieve an even more accurate optimised pose graph and global map.

Next we presented occupancy grid maps as our choice of map representation. A detailed explanation was given on the derivation of the occupancy grid mapping algorithm as well as the consequential assumption of conditional independence between cells within the map. An introduction to ray casting and *OctoMap* was also given, which outlined the basis of implementing a sensor model to deal with uncertainty in measurements. This preceded a detailed discussion of two inverse sensor models (ISMs). The first presented ISM was one detailed by Thrun where a derivation on the model was given accompanied by a discussion on the effect of varying parameters within the model. We found that very little information was available on the choice of the L value which is responsible for acting as a buffer value for measurements that do not fall within the centre of a cell of the occupancy grid map. Literature indicated that the choice of the L value was recommended as using the diagonal length of the cell width (resolution), while some researchers chose an arbitrary value with little explanation. This therefore lead to an investigation on the effects that the selection of this value has on the resulting map. The second presented ISM was one detailed by Andert, although information on a derivation for the model proved to be lacking in literature, which only allowed for a detailed description of the various components of the algorithm to be made. While Andert's model was originally designed for stereo vision, a small adaptation was made to accommodate monocular vision since we now no longer needed to take the quadratic error faced by stereo vision into account. This was again followed by a description of the various parameters that effect the resulting probability of a measurement. At this stage we also found that the significance factor k, which is responsible for assigning a weighting to a measurement to determine its impact on the resulting map, was also lacking guidance, in literature, for the choice of the value. Therefore, similarly to Thrun's L value, this lead to an investigation on the effects of varying the significance factor for Andert's ISM.

At this point we had the fundamentals in place to create a system that used monocular sensor data through LSD-SLAM to apply semi-dense depth estimates to one of two inverse sensor models (Thrun and Andert). The ISM's were used to determine the probabilities of occupancy for each measurement and inserted them into an occupancy grid map. Each cell within the map was updated using log odds notation to improve the efficiency of cell updates with addition of log odds, as opposed to multiplication of probabilities. The system ran within the Robotics Operating System (ROS) environment which enabled us to create an efficient test platform.

In Part 1 of our tests, a methodology of the experiments was detailed. We started with an overview of the project, followed by a critical discussion on available datasets for our experiments. We highlighted a clear shortage of available datasets for our experiments as LSD-SLAM was either too sensitive to the camera movements, or perfect ground truth was not available. This was a crucial observation, specifically for our tests on accuracy, as we were not aiming to measure the the accuracy of the resulting *trajectory* of LSD-SLAM, but rather the accuracy of the resulting *map* of our system. Along with our investigation on datasets, we highlighted some datasets that would be well suited for our remaining tests on memory, performance and qualitative analysis, since a ground truth was not necessary for these criteria. This part of the thesis was also used to discuss our aims and methodology for performing these experiments on the identified feasible datasets.

In Part 2 of our tests we showed the results achieved for each of the evaluation criteria.

Memory: Memory tests were performed on 5 resolutions using 2 methods of map storage, one which stored the map as a maximum likelihood binary state, and one which preserved all probabilistic information. We also performed the tests on both ISM's on 2 separate datasets (a large outdoor and small indoor dataset). We attempted to evaluate not only the file size of the resulting maps, but also the on-board RAM usage of each resulting map. We found that, as expected, higher resolution maps (such as 8 mm or 16 mm resolutions) for large environments are not feasible for any of the tested ISM's, as the RAM usage easily exceeds 10 Gb; however, it does prove to be feasible on lower resolutions. The large environment dataset also highlighted the difference between Thrun and Andert's ISM in terms of the effect these models have on the ability for the map to be compressed efficiently. Andert's ISM was identified to create a larger and uneven distribution of probabilities within a map when compared to Thrun. This causes a lower likelihood for cells with equal probabilities and between clamping thresholds to exist, which would most often occur at the clamping thresholds. The smaller dataset did not cause notable concerns in terms of file size or RAM consumption. In terms of file type, for either dataset, the binary state always created a smaller file than the full probability state, however the RAM usage remained very similar for either storage method.

Performance: To evaluate the performance of our system with LSD-SLAM, we used both a large and small environment dataset again. The average time taken to process and update 100 k depth estimates from LSD-SLAM was calculated, while LSD-SLAM was running in parallel. We found that for both datasets, Thrun's model performed significantly faster than Andert's model across all 5 resolutions tested. We also showed that the size of the dataset (or environment) has a large impact on the performance of the process time per 100 k points, however lowering the resolution does mitigate this problem.

Accuracy: From a purely visual perspective, some results were presented to get a better intuitive understanding of how differently the ISM's would perform on different datasets. The quality of the maps were evaluated on the office scene dataset by selecting 3 parameter values for each of the ISM (both Thrun and Andert's ISM) at a resolution of 16 mm. We showed that for Thrun's ISM, the buffer value L has a large effect on the generated map, where an increase in L corresponds to wider predictions on occupancy. We also showed that for Andert's ISM, the significance factor k, did not necessarily cause wider predictions on occupancy, but rather caused the predictions of occupancy to be more prominent.

Our most important tests performed in this thesis were quantitative evaluations on accuracy between the ISM's on a synthetic dataset. A total of 100 maps were created to cater for both ISM's, with 10 parameter value changes each, at 5 resolutions. Each occupancy grid map was created using our system, and then scaled and transformed using minimisation techniques to match the ground truth map. The maps were then divided into slices along the height, with each slice converted into a gray scale image to represent the probabilistic value as a shade of gray for *free*, *unknown* and *occupied* space from *black* to *white* respectively. A ROC curve analysis was run on every map across every image on a pixel by pixel basis to compare the the state of the cell against a contingency table. Each comparison was made based on a threshold of occupancy, which spanned many thresholds between 0 and 1.

The resulting ROC curves suggested that Thrun's model performed the best by achieving the highest area under the curve (AUC) of 0.91 with an L value of 0.5 at a resolution of 8 mm. Thrun's model always seemed to achieve the best accuracy when an L value of between 0.4 and 0.5 was chosen, regardless of the resolution used. These results showed that the *standard* way of selecting the L value by taking the diagonal length of a cell is not necessarily a good choice. Andert's model showed an interesting response due to the effect of using a static minimum probability assignment for a *free* cell. Andert's model created a different probability distribution pattern compared to Thrun. Thrun's model created a map with a smaller distribution of probabilities between clamping thresholds with the majority of the cells *at* the clamping thresholds, while Andert's model had more probabilities between the clamping thresholds with peaks of frequency at the clamping thresholds as well as on the minimum probability assignment value (0.3 in our case). It was also found that the choice of the significance factor k for Andert's model, which is responsible for dictating the significance of a measurement within the map, is highly dependent on the resolution of the map. In other words, the choice of the k value depends on the preference of how many measurements are required to have an occupied cell reach the maximum probability of occupancy (clamping threshold).

Based on our investigation and experiments, we can conclude that monocular vision and LSD-SLAM can indeed be used to create an accurate semi-dense occupancy grid map to be used for autonomous navigation. Using Thrun's inverse sensor model may provide very accurate maps, and when looking at the insertion times of Thrun's ISM, and the average number of points processed per key frame for the tested dataset, real time performance of 30 Hz is possible. However, this level of performance is dependent on the speed at which LSD-SLAM processes individual frames and key frames. Our results show that for the open-source version of LSD-SLAM, and on default settings, image processing at 30 Hz is not possible. We also found that LSD-SLAM is not as robust as the authors claim, where many of the tested datasets failed to be tracked for any extended period of time.

Realistically, the autonomous platform using the system presented in this thesis would need to move at slower speeds and at stable trajectories to achieve accurate results. The platform would also require the hardware to utilise at least 4 cores of processing power, as well as at least 4 Gb of RAM. It would appear that LSD-SLAM is the weak link within this system, although if the process time were improved, the resulting system would be perfectly viable for autonomous flying systems, rather than likely being limited to ground based platforms.

The tests performed were useful in determining certain characteristics and it is unfair to use only those scores as the determining factor of the usefulness of the resulting map. A map that represents some portion or sparse version of the environment may score low, even if a path planning algorithm can calculate a navigable path within the environment using the map. Such a map can therefore be considered useful.

7.1.1 Contributions

- **ISM Comparison** In this thesis we showed an in depth comparison between two very well known and widely used inverse sensor models for use on monocular vision semi-dense data. We identified a feature of each of the ISMs that was not associated much discussion in literature, and showed how they effected the resulting occupancy grid maps. We also presented a better understanding as to why the two ISMs perform the way they do.

- **Semi-dense** We also showed the viability of using semi-dense monocular depth data as a means for creating an occupancy grid map for autonomous navigation and obstacle avoidance. If a map is incomplete it does not mean it is not useful to an a path planning algorithm or an autonomously navigating platform. We showed that the semi-dense data output by LSD-SLAM could produce accurate occupancy grid maps when applied to the appropriate inverse sensor model

- **Open-Source** Another contribution of this work was to create an open-source platform for more research to be carried out on a working system, even though complete real-time applications may not be feasible. Our mapping system is not optimised, and further research is continuously being done on LSD-SLAM, which creates a large opportunity to apply LSD-SLAM (or its variants) along with our mapping system to a real world platform.
- **Datasets** Finally, we also identified that there is a lack of monocular vision based datasets with perfect ground truth maps of both large and small scale environments.

7.2 Future Work

As mentioned in Chapter 1, one disadvantage of monocular vision is scale ambiguity, although one could employ a form of sensor fusion with our system to add a sense of scale. One route could be to add an IMU (inertial measurement unit) that, if calibrated correctly, could help determine scale of trajectory to translate into scale of mapping. It could also be combined with pose and depth estimates on LSD-SLAM to improve not only tracking, but image alignment between key frames. Another method of sensor fusion could be a station for depth seeding to initialise the correct scale of the map. For example, a robot could be placed on a docking station with a dense RGB-D sensor such as a Microsoft Kinect, which translates the depth of the scene to the LSD-SLAM algorithm. The initial depth image can be used as a depth seed to propagate through the map as the robot then explores free of the docking station.

In Chapter 6 we mentioned that our system is not optimised. Future work could be done to improve the performance of not only the occupancy grid mapping algorithm, but also portions of LSD-SLAM's algorithm in both tracking and mapping. More benefit may be gained by improving the robustness and speed of tracking components of LSD-SLAM.

One could also improve the way LSD-SLAM works by combining its direct pixel tracking method with feature based methods. The load could be split into using features for tracking and direct image alignment for depth estimation.

The potential for this system is enormous when considering the growing trend of autonomous vehicles on the roads today, and the databases of video feed received by them. As monocular vision, massive scale occupancy grid map can be created to share amongst autonomous cars to help them navigate better. And since they are generally confined to roads, which are a highly structure environment, many features (such as traffic lights or signs) could be extracted to incorporate the correct scale into the map.

While this thesis is focused on the accuracy and viability of using LSD-SLAM for mapping or autonomous navigation, the next logical step is to demonstrate the usefulness of the created map. This can be done by determining false positive and false negative *paths* that are estimated by a *path planning* or *obstacle avoidance* algorithm. False positives, in the context of path planning, is when an obstacle free path is created based on a ground truth map, but is considered to be blocked by an obstacle when applied to the generated map. False negative paths are when a path created based on the generated map produces an invalid path when applied to the ground truth map. Applying these two metrics to both the ground truth map and the generated map forms the basis for path planning based evaluation [79]. Our system allows for a local map of any size to be created while LSD-SLAM runs on its own processing cores. The local map could be limited to map only a few meters around the current pose, which allows for obstacle avoidance within a map that has yet to perform loop closure and that is experiencing drift.

We believe that the work done in this thesis has made some contribution towards the the goal of developing an autonomously navigating platform using monocular vision.

List of References

[1] Durrant-Whyte, H. and Bailey, T.: Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE RAM*, 2006.

[2] Hartley, R. and Zisserman, A.: *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[3] Klein, G. and Murray, D.: Parallel tracking and mapping for small ar workspaces. In: *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 225–234. IEEE, 2007.

[4] Engel, J., Schöps, T. and Cremers, D.: Lsd-slam: Large-scale direct monocular slam. In: *European Conference on Computer Vision*, pp. 834–849. Springer, 2014.

[5] Newcombe, R.A., Lovegrove, S.J. and Davison, A.J.: Dtam: Dense tracking and mapping in real-time. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2320–2327. IEEE, 2011.

[6] Filliat, D.: Loop closure detection. 2004.

[7] Engel, J.: Semi-dense slam. December 2015.

[8] www8.cs.umu.se: Path planning. December 2015.

[9] Tang, R.: Custom player plugins. September 2017.

[10] Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C. and Burgard, W.: Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[11] Thrun, S., Burgard, W. and Fox, D.: *Probabilistic robotics.* MIT press, 2005.

[12] Hobbyzine: Robot obstacle detection and avoidance with the devantech srf05 ultrasonic range finder. nd. Accessed: 2017-05-5.

[13] Wasenmüller, O., Meyer, M. and Stricker, D.: CoRBS: Comprehensive rgb-d benchmark for slam using kinect v2. In: *IEEE Winter Conference on Applications of Computer Vision (WACV)*, vol. ., p. . IEEE, March 2016. ISBN . Available at: `http://corbs.dfki.uni-kl.de/`

[14] Zhou, Q.-Y., Miller, S. and Koltun, V.: Elastic fragments for dense scene reconstruction. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 473–480. 2013.

[15] Handa, A., Whelan, T., McDonald, J. and Davison, A.J.: A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In: *Robotics and automation (ICRA), 2014 IEEE international conference on*, pp. 1524–1531. IEEE, 2014.

[16] Nordhaus, W.D.: The progress of computing. *Cowles Foundation Discussion Paper No. 1324*, September 2001.

[17] Koh, H. and Magee, C.L.: A functional approach for studying technological progress: Application to information technology. *Technological Forecasting and Social Change*, vol. 73, no. 9, pp. 1061–1083, 2006.

[18] Impacts, A.: Trends in the cost of computing. March 2015.

[19] Cadena, C.: Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. In: *?* 2016.

[20] Frese, U.: Interview: Is slam solved? *KI-Künstliche Intelligenz*, vol. 24, no. 3, pp. 255–257, 2010.

[21] Chong, T.: Sensor technologies and simultaneous localization and mapping (slam). In: *Procedia Computer Science*. 2015.

[22] Nï¿½tzi, G.: *Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM*. Master's thesis, ETH Zurich, 2010.

[23] Song, S. and Chandraker, M.: Robust scale estimation in real-time monocular sfm for autonomous driving. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1566–1573. 2014.

[24] Engel, J., Stückler, J. and Cremers, D.: Large-scale direct slam with stereo cameras. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 1935–1942. IEEE, 2015.

[25] Engel, J.: Lsd-slam: Large-scale direct monocular slam. 2016.
Available at: `http://vision.in.tum.de/research/vslam/lsdslam`

[26] Jack, A.: *Semi-Dense and Dense 3D Scene Understanding in Embedded Multicore Processors*. Master's thesis, Imperial College London, 2015.

[27] Joubert, D.: *Adaptive occupancy grid mapping with measurement and pose uncertainty*. Master's thesis, Stellenbosch University, 2012.

[28] Thrun, S.: Learning occupancy grid maps with forward sensor models. *Autonomous robots*, vol. 15, no. 2, pp. 111–127, 2003.

[29] Yamauchi, B.: A frontier-based approach for autonomous exploration. In: *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pp. 146–151. IEEE, 1997.

[30] Thrun, S.: Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[31] Burgard, W., Moors, M., Fox, D., Simmons, R. and Thrun, S.: Collaborative multi-robot exploration. In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 476–481. IEEE, 2000.

[32] Simmons, R., Apfelbaum, D., Burgard, W., Fox, D., Moors, M., Thrun, S. and Younes, H.: Coordination for multi-robot exploration and mapping. In: *AAAI/IAAI*, pp. 852–858. 2000.

[33] Torabi, L., Kazemi, M. and Gupta, K.: Configuration space based efficient view planning and exploration with occupancy grids. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2827–2832. IEEE, 2007.

[34] Pagac, D., Nebot, E.M. and Durrant-Whyte, H.: An evidential approach to map-building for autonomous vehicles. *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 623–629, 1998.

[35] Moravec, H. and Elfes, A.: High resolution maps from wide angle sonar. In: *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, pp. 116–121. IEEE, 1985.

[36] Burger, A.J.: *Occupancy grid mapping using stereo vision.* Ph.D. thesis, Stellenbosch: Stellenbosch University, 2015.

[37] Gutmann, J.-S., Fukuchi, M. and Fujita, M.: 3d perception and environment map generation for humanoid robot navigation. *The International Journal of Robotics Research*, vol. 27, no. 10, pp. 1117–1134, 2008.

[38] Elinas, P., Hoey, J., Lahey, D., Montgomery, J.D., Murray, D., Se, S. and Little, J.J.: Waiting with josé, a vision-based mobile robot. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 4, pp. 3698–3705. IEEE, 2002.

[39] Borenstein, J. and Koren, Y.: The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[40] Čikeš, M., Đakulović, M. and Petrović, I.: The path planning algorithms for a mobile robot based on the occupancy grid map of the environment?a comparative study. In: *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*, pp. 1–8. IEEE, 2011.

[41] Simmons, R., Goodwin, R., Koenig, S., O'Sullivan, J. and Armstrong, G.: 5 xavier: An autonomous mobile robot on the web. *Beyond Webcams: an introduction to online robots*, p. 81, 2002.

[42] Müller, J.: Autonomous navigation for miniature indoor airships. *Diss. Universitätsbibliothek Freiburg*, 2013.

[43] Maier, D., Lutz, C. and Bennewitz, M.: Integrated perception, mapping, and footstep planning for humanoid navigation among 3d obstacles. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2658–2664. IEEE, 2013.

[44] Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A.Y.: Ros: an open-source robot operating system. In: *ICRA Workshop on Open Source Software*. 2009.

[45] Lobo, J., Marques, L., Dias, J., Nunes, U. and de Almeida, A.T.: Sensors for mobile robot navigation. In: *Autonomous Robotic Systems*, pp. 50–81. Springer, 1998.

[46] Adarsh, S., Kaleemuddin, S.M., Bose, D. and Ramachandran, K.: Performance comparison of infrared and ultrasonic sensors for obstacles of different materials in vehicle/robot navigation applications. In: *IOP Conference Series: Materials Science and Engineering*, vol. 149, p. 012141. IOP Publishing, 2016.

[47] Modi, S.: *Comparison of three obstacle avoidance methods for an autonomous guided vehicle*. Ph.D. thesis, College of Engineering 2002 by Sachin Modi BE (Mechanical Engineering) Manarashtra Institute of Technology, University of Pune, India, 1998.

[48] Brink, W.: *Stereo vision for simultaneous localization and mapping*. Ph.D. thesis, Stellenbosch: Stellenbosch University, 2012.

[49] Sola, J.: Simulataneous localization and mapping with the extended kalman filter. *unpublished. Available: http://www. joansola. eu/JoanSola/eng/JoanSola. html*, 2013.

[50] Kim, J.-H., Yang, W., Jo, J., Sincak, P. and Myung, H.: *Robot Intelligence Technology and Applications 3: Results from the 3rd International Conference on Robot Intelligence Technology and Applications*, vol. 345. Springer, 2015.

[51] Davison, A.J., Reid, I.D., Molton, N.D. and Stasse, O.: Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, 2007.

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

[53] Koenderink, J.J. and Van Doorn, A.J.: Affine structure from motion. *JOSA A*, vol. 8, no. 2, pp. 377–385, 1991.

[54] Zhang, Z.: Microsoft kinect sensor and its effect. *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[55] Strasdat, H., Montiel, J. and Davison, A.J.: Scale drift-aware large scale monocular slam. *Robotics: Science and Systems VI*, 2010.

[56] Eade, E. and Drummond, T.: Edge landmarks in monocular slam. In: *BMVC*, pp. 7–16. 2006.

[57] Concha, A. and Civera, J.: Using superpixels in monocular slam. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 365–372. IEEE, 2014.

[58] Engel, J., Sturm, J. and Cremers, D.: Semi-dense visual odometry for a monocular camera. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1449–1456. 2013.

[59] Glover, A., Maddern, W., Warren, M., Reid, S., Milford, M. and Wyeth, G.: Openfabmap: An open source toolbox for appearance-based loop closure detection. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 4730–4735. IEEE, 2012.

[60] Choset, H.M.: *Principles of robot motion: theory, algorithms, and implementation.* MIT press, 2005.

[61] Yang, L., Qi, J., Song, D., Xiao, J., Han, J. and Xia, Y.: Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, vol. 2016, p. 5, 2016.

[62] LaValle, S.M.: *Planning algorithms.* Cambridge university press, 2006.

[63] Meyer-Delius, D., Beinhofer, M. and Burgard, W.: Occupancy grid models for robot mapping in changing environments. In: *AAAI*. 2012.

[64] Aghababa, M.P.: 3d path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles. *Applied Ocean Research*, vol. 38, pp. 48–62, 2012.

[65] Fakoor, M., Kosari, A. and Jafarzadeh, M.: Humanoid robot path planning with fuzzy markov decision processes. *Journal of Applied Research and Technology*, vol. 14, no. 5, pp. 300–310, 2016.

[66] Sebastian Säger, A.P.: Lsd-slam evaluation and extension. In: *Computer Vision: Object and People Tracking*, p. 74. Department Augmented Vision - University of Kaiserslautern and DFKI GmbH, Oliver Wasenmüller, Didier Stricker, March 2015.

[67] Krombach, N., Droeschel, D. and Behnke, S.: Combining feature-based and direct methods for semi-dense real-time stereo visual odometry. In: *International Conference on Intelligent Autonomous Systems*. 2016.

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

[69] J. Engel, T. Schï¿½ps, D.C.: Lsd-slam: Large-scale direct monocular slam. March 2014. Accessed: 2017-03-10.

[70] Yguel, M., Aycard, O. and Laugier, C.: Update policy of dense maps: Efficient algorithms and sparse representation. In: *Field and Service Robotics*, pp. 23–33. Springer, 2008.

[71] Gallup, D., Frahm, J.-M., Mordohai, P. and Pollefeys, M.: Variable baseline/resolution stereo. In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.

[72] Einhorn, E., Schröter, C. and Gross, H.-M.: Finding the adequate resolution for grid mapping-cell sizes locally adapting on-the-fly. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1843–1848. IEEE, 2011.

[73] Kai M. Wurm, A.H.: Octomap: An efficient probabilistic 3d mapping framework based on octrees. March 2017. Accessed: 2017-03-20.

[74] Pathak, K., Birk, A., Poppinga, J. and Schwertfeger, S.: 3d forward sensor modeling and application to occupancy grid based sensor fusion. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 2059–2064. IEEE, 2007.

[75] Elfes, A.: Occupancy grids: A stochastic spatial representation for active robot perception. In: *Proceedings of the Sixth Conference on Uncertainty in AI*, vol. 2929. 1990.

[76] Andert, F.: Drawing stereo disparity images into occupancy grids: Measurement model and fast implementation. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 5191–5197. IEEE, 2009.

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

[78] Baldi, P., Brunak, S., Chauvin, Y., Andersen, C.A. and Nielsen, H.: Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.

[79] Colleens, T. and Colleens, J.: Occupancy grid mapping: An empirical evaluation. In: *Control & Automation, 2007. MED'07. Mediterranean Conference on*, pp. 1–6. IEEE, 2007.

[80] Marzat, J., Moras, J., Plyer, A., Eudes, A. and Morin, P.: Vision-based localization, mapping and control for autonomous mav: Euroc challenge results. In: *15th ONERA-DLR Aerospace Symposium (ODAS 2015)*. 2015.

[81] Balaguer, B., Balakirsky, S., Carpin, S. and Visser, A.: Evaluating maps produced by urban search and rescue robots: lessons learned from robocup. *Autonomous Robots*, vol. 27, no. 4, p. 449, 2009.

[82] Pestana, J., Prettenthaler, R., Holzmann, T., Muschick, D., Mostengel, C., Fraundorfer, F. and Bischof, H.: Graz griffins solution to the european robotics challenges 2014. In: *Proceedings of Austrian Robotics Workshop 2015*, pp. 11–12. 2015.

[83] Holzmann, T., Prettenthaler, R., Pestana, J., Muschick, D., Graber, G., Mostegel, C., Fraundorfer, F. and Bischof, H.: Performance evaluation of vision-based algorithms for mavs. *arXiv preprint arXiv:1505.02247*, 2015.

[84] Matthews, B.W.: Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[85] Ding, Z.: Diversified ensemble classifiers for highly imbalanced data learning and their application in bioinformatics. 2011.

[86] Vihinen, M.: How to evaluate performance of prediction methods? measures and their interpretation in variation effect analysis. *BMC genomics*, vol. 13, no. 4, p. S2, 2012.

[87] Schöps, T., Engel, J. and Cremers, D.: Semi-dense visual odometry for ar on a smartphone. In: *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pp. 145–150. IEEE, 2014.