Building dense reconstructions with SLAM and Spot

Ola Nilsson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6158
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2022

Abstract

Having access to dense reconstruction of ongoing building constructions provides insight into the building process and could serve as both a tool for error detection and documentation of the actual outcome. In this thesis, Spot, the quadruped robot designed by Boston Dynamics is evaluated as a platform for site inspection. The result of the thesis was a prototype system built on top of the ROS framework which integrates the sensors of the robot platform. Combined with the visual SLAM library RTABMap for robust localization with loop closing capabilities and dense camera and LiDAR reconstructions, the system provides means for online map building and data acquisition. The localization performance achieved positional RMSE in the decimeter range for paths spanning further than 100 meters. The resulting LiDAR reconstructions provided comparable results to be evaluated against the planned building model and acheived centimeter accuracy with the current sensor configuration.

Acknowledgements

This thesis was carried out as a part of the following research projects:

- SBUF "Utmätning- & utsättningsrobotar– Ökad precision och informationsåterkoppling på byggarbetsplatsen"
- VINNOVA (UDI-2) "Innovativ Agile Construction for globally improved sustainability" (ACon 4.0)
- SSF "Semantic Mapping and Visual Navigation for Smart Robots" (RIT15-0038)

I would like to express my gratitude to all the people involved who made the project possible and contributed to its betterment. First off, I would like to thank Anders Robertsson (Dept. of Automatic Control, LTH), Mathias Haage (Dept. Computer Science, LTH), Helena Eriksson (Cognibotics), and Greg Austin for their valuable input, encouragement and support during the project. I want to thank Jonas Hansson (Dept. of Automatic Control) for helping me acquire the ground truth measurements used in the evaluation of the localization performance. I would also like to thank Anton Tetov Johansson (Dept. of Automatic Control, LTH) for providing the necessary means to sample the building information model. Finally, I want to thank my significant other Hedda Magnusson, not only for her meticulous proofreading of the report, but also for her unending patience and support.

Contents

1

Introduction

The use of mobile robots for inspection is increasing as advancements in the field of mobile exploration are made. The mobile exploration field is multi-disciplinary and the knowledge involved combines multiple research areas, such as robotics, computer vision and optimization. Simultaneous localization and mapping is a heavily researched topic within the mobile exploration field which combines all the previously mentioned disciplines. Simultaneous Localization and Mapping, abbreviated SLAM, solves the problem of estimating the pose of a robot in an environment while simultaneously building a map of said environment. SLAM also allows for navigation in scenarios where GPS tracking is unavailable.

With the design of quadruped robots such as *Spot*[1] that show remarkable mobility and are capable of traversing rough terrain and dynamic environments in which conventional mobile robots with tracks or wheels are not, interesting use cases are made possible by combining a mobile platform such as Spot with previously mentioned mapping and localization techniques.

For instance, in the construction field miscommunication and human error eventually leads to construction errors which are costly in terms of both monetary loss and time. In 2018, the Swedish authority Boverket published a report surveying faults, deficiencies and damages in the construction sector. The annual costs associated with faulty construction were estimated to exceed 80 billion SEK [Boverket, 2018]. A mobile robot such as Spot could be utilized for site inspection and building dense reconstructions with SLAM. Having access to dense reconstructions of the ongoing construction could provide means for detecting and preventing construction errors. By finding and correcting errors early on in the construction process the monetary loss can be reduced. In addition, the reconstructions may also be used as documentation of deviations by comparing the planned model or blueprint to as-built.

[1] https://www.bostondynamics.com/products/spot Accessed:2022-03-26

1.1 Objectives

This thesis was carried out in collaboration with the Center for Construction Robotics at Lunds Tekniska Högskola. The aim of the thesis was to evaluate the feasability of creating dense reconstructions with the available sensor configuration on Spot. The main objectives of the thesis were;

- Build a prototype visual SLAM system, which incorporates the available sensor configuration on Spot.
- Benchmark the localization performance of the system by comparing the pose estimate output from the SLAM system against ground truth measurements.
- Evaluate the accuracy of the reconstructions by comparing the reconstructions with the building information model.

1.2 Outline

- *Chapter 2* presents the theory necessary to understand the SLAM problem and performance metrics, methods for aligning point clouds and the implementation.
- *Chapter 3* presents Spot and goes into details regarding the implementation. The chapter is concluded with the results from a working prototype system and the initial impressions.
- *Chapter 4* presents the results of the evaluation of the localization performance and the methods for evaluating the reconstructions against a planned building model.
- *Chapter 5* presents the reflections regarding the results.
- *Chapter 6* concludes the report with a brief summary, final remarks and possible further development.

2

Theory

This chapter presents the theoretical foundation of which this thesis builds upon. The chapter begins by describing homogenous transformations and is then succeeded by some fundamental computer vision topics. Continuing, the SLAM problem and the different paradigms are presented. Finally, the absolute orientation problem which addresses the problem of aligning 3D point correspondences is described.

2.1 Homogenous transformations

Homogenous transformations are useful for representing the position and orientation of a rigid body as well as changing the reference frame of the representation of a point or vector. A homogenous transformation $\mathbf{H} \in SE(3)$ is a 4×4 matrix that

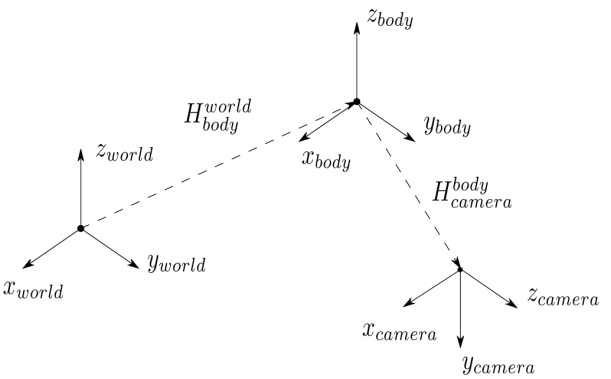

Figure 2.1 Relating coordinate frames with homogenous transformations.

encodes rotation and translation in a compact representation such that

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.1}$$

where $\mathbf{R} \in SO(3)$ is a 3×3 rotation matrix with $\det(\mathbf{R}) = 1$, and $\mathbf{t} \in \mathbb{R}^3$ a translation vector. The homogenous transformation has certain properties. $\mathbf{H}$ is invertible such that

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.2}$$

Given two homogenous transformations $\mathbf{H}_1$ and $\mathbf{H}_2$, the resulting matrix from the product of the transformations $\mathbf{H}_1$ and $\mathbf{H}_2$ is also a homogenous transformation $\mathbf{H}_3 \in SE(3)$. The multiplication of homogenous matrices are associative, $\mathbf{H}_1(\mathbf{H}_2\mathbf{H}_3) = (\mathbf{H}_1\mathbf{H}_2)\mathbf{H}_3$, but generally not commutative, $\mathbf{H}_1\mathbf{H}_2 \neq \mathbf{H}_2\mathbf{H}_1$ [Lynch and Park, 2017].

The mapping of a point acquired in a reference frame b can be transformed into the reference frame a given the homogenous transformation $\mathbf{H}_b^a$. A small example for a mobile robot with a camera rigidly attached to the robot body is illustrated in Figure 2.1. For a point $\mathbf{p}$ with homogenous coordinates given in the camera frame, the representation of $\mathbf{p}$ in the world frame is then given by

$$\mathbf{p}_{world} = \mathbf{H}_{body}^{world}\mathbf{H}_{camera}^{body}\mathbf{p} \quad = \mathbf{H}_{camera}^{world}\mathbf{p}. \tag{2.3}$$

2.2 Computer Vision

This section describes computer vision topics that are relevant for understanding the concepts of camera motion which is a core concept to understand the visual SLAM problem. The theory presenting the camera model, epipolar geometry and depth from stereo views is based on the lecture notes [Olsson, 2021].

Pinhole camera model

The pinhole camera model is the simplest camera model. The projection $\mathbf{x} = (x_1, x_2, 1)^T$ of a 3D scene point $\mathbf{X} = (X_1, X_2, X_3)^T$ onto the image plane is done by forming a line between $\mathbf{X}$ and the camera center $\mathbf{C}$ which intersects the image plane at $z_{camera} = 1$. A drawing of the model description is shown in Figure 2.2. The normal to the image plane is the principal axis of the camera.

The direction of the viewing ray $\mathbf{l}_{ray}$ may be parameterized by

$$\mathbf{l}_{ray} = \mathbf{C} + s(\mathbf{X} + \mathbf{C}), \quad s \in \mathbb{R} \tag{2.4}$$

For simplicity, placing the camera center at the origin $\mathbf{C} = (0,0,0)^T$ reduces the expression in (2.4) to

$$\mathbf{l}_{ray} = s\mathbf{X}, \quad s \in \mathbb{R} \tag{2.5}$$

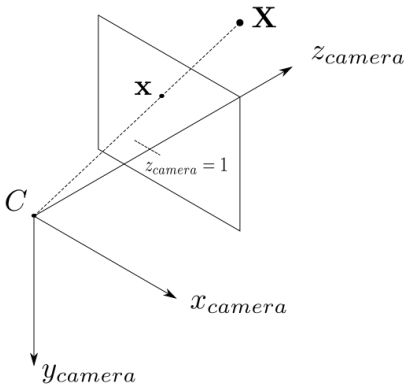

Figure 2.2 The pinhole camera model.

The intersection with the image plane is found by computing s such that $sX_3 = 1$. This results in

$$\mathbf{x} = \begin{bmatrix} X_1/X_3 \\ X_2/X_3 \\ sX_3 \end{bmatrix} = \begin{bmatrix} X_1/X_3 \\ X_2/X_3 \\ 1 \end{bmatrix}, \quad X_3 \neq 0 \tag{2.6}$$

The projective mapping from a point in the scene to the image coordinate frame may be described by the 3×4 camera matrix $\mathbf{P}$

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tag{2.7}$$

where

$$\mathbf{K} = \begin{bmatrix} \gamma f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

is the camera intrinsic matrix which maps points on the image plane to pixel coordinates. γ, f, and s denote the aspect ratio, focal length and skew, respectively. (x_0, y_0) is typically referred to as the optical center or principal point. $[\mathbf{R} \quad \mathbf{t}]$ are the camera extrinsics which describe the relative rotation and translation of the camera to a refence frame. The projection may now be described by

$$\mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X} = \mathbf{P}\mathbf{X} \tag{2.9}$$

Epipolar geometry

The problem of recovering both 3D points and camera matrices is referred to as the two-view structure from motion problem [Olsson, 2021]. Given two sets of point

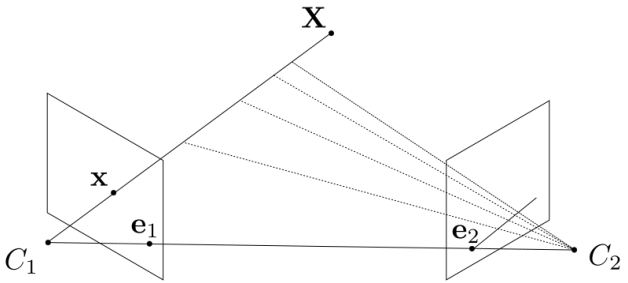

Figure 2.3 An illustration of the geometry of the epipolar constraints, inspired by the figure in [Olsson, 2021].

correspondances $\mathbf{x}$ and $\tilde{\mathbf{x}}$, both 3D points and the relative camera motion may be recovered through exploiting the epipolar geometry of the problem. Figure 2.3 illustrates the setup of the problem.

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be the camera matrices with camera centers $\mathbf{C}_1$ and $\mathbf{C}_2$ respectively. The viewing ray $\mathbf{l}$ resulting from the projection of a 3D point $\mathbf{X}$ into camera $\mathbf{P}_1$ can be parameterized as

$$\mathbf{X}(\lambda) = \begin{bmatrix} \lambda \mathbf{x} \\ 1 \end{bmatrix} \tag{2.10}$$

The projection of (2.10) into camera $\mathbf{P}_2$ is then described by

$$\mathbf{P}_2\mathbf{A}(\lambda) = \begin{bmatrix} \mathbf{A} & \mathbf{t} \end{bmatrix} \begin{bmatrix} \lambda \mathbf{x} \\ 1 \end{bmatrix} \quad = \lambda \mathbf{A}\mathbf{x} + \mathbf{t} \tag{2.11}$$

The line in (2.11) is referred to as an epipolar line. All points $\mathbf{X}$ that project to $\mathbf{x}$ must lie on this line. This forms the epipolar constraint. The epipoles $\mathbf{e}_1$ and $\mathbf{e}_2$ are the projections of the camera centers into the cameras, meaning

$$\begin{aligned} \mathbf{e}_1 &= \mathbf{P}_1\mathbf{C}_2 \\ \mathbf{e}_2 &= \mathbf{P}_2\mathbf{C}_1 \end{aligned} \tag{2.12}$$

Now, since all viewing rays $\mathbf{l}$ formed by projections into $\mathbf{P}_1$ intersect $\mathbf{C}_1$ from (2.4) it follows that all epipolar lines will intersect in $\mathbf{e}_2$. From the parametrization of the epipolar line in (2.11) letting $\lambda = 0$ results in

$$\mathbf{e}_2 = \mathbf{t} \tag{2.13}$$

$$\begin{cases} \mathbf{l}^T\mathbf{t} = 0 \\ \mathbf{l}^T(\mathbf{A}\mathbf{x} + \mathbf{t}) = 0 \end{cases} \tag{2.14}$$

Since $\mathbf{l}$ is perpendicular to both $\mathbf{t}$ and $\mathbf{Ax}+\mathbf{t}$ it follows that

$$\mathbf{l} = \mathbf{t} \times (\mathbf{Ax}+\mathbf{t}) \quad = \mathbf{t} \times \mathbf{Ax} \tag{2.15}$$

Inserting (2.13) into (2.15) we obtain

$$\mathbf{l} = \mathbf{e}_2 \times \mathbf{Ax} \quad = [\mathbf{e}_2]_\times \mathbf{Ax} \tag{2.16}$$

Now we can define the fundamental matrix

$$\mathbf{F} = [\mathbf{e}_2]_\times \mathbf{A} \tag{2.17}$$

which maps a point $\mathbf{x}$ in image 1 to lines in image 2. If $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are point correspondances then the epipolar constraint may be expressed as

$$\tilde{\mathbf{x}}\mathbf{F}\mathbf{x} = 0 \tag{2.18}$$

The camera $\mathbf{P}_2$ can now be extracted from $\mathbf{F}$

$$\mathbf{P}_2 = \begin{bmatrix} [\mathbf{e}_2]_\times \mathbf{F} & \mathbf{e}_2 \end{bmatrix} \tag{2.19}$$

Depth recovery from stereo views

For a camera pair with known camera matrices, the depth can be estimated for each pixel in an image where pixel matches can be found in both views. Combining the results from Section 2.2 and the fact that the camera matrices are known, the search for pixel matches can be constrained to searching along the epipolar line.

For the special case of a stereo camera, the two views are rigidly coupled and separated by a translation along the x-axis such that $\mathbf{t} = (b,0,0)^T$ with camera matrices

$$\begin{aligned} \mathbf{P}_1 &= \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \\ \mathbf{P}_2 &= \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix} \end{aligned} \tag{2.20}$$

Figure 2.4 shows an illustration of the setup. The projection of a 3D point $\mathbf{X}$ into both views results in

$$\begin{aligned} \mathbf{x}_1 = \mathbf{P}_1\mathbf{X} = & \begin{bmatrix} \gamma f X_1 + s f X_2 + x_0 X_3 \\ f X_2 + y_0 X_3 \\ X_3 \end{bmatrix} \\ \mathbf{x}_2 = \mathbf{P}_2\mathbf{X} = & \begin{bmatrix} \gamma f (X_1 + b) + s f X_2 + x_0 X_3 \\ f X_2 + y_0 X_3 \\ X_3 \end{bmatrix}. \end{aligned} \tag{2.21}$$

From Equation 2.21 it is shown that the y-component of the pixel value for both projections are equal for any corresponding point. It then follows that the epipolar lines for the setup are horizontal e.g., parallell to the x-axis. The difference between

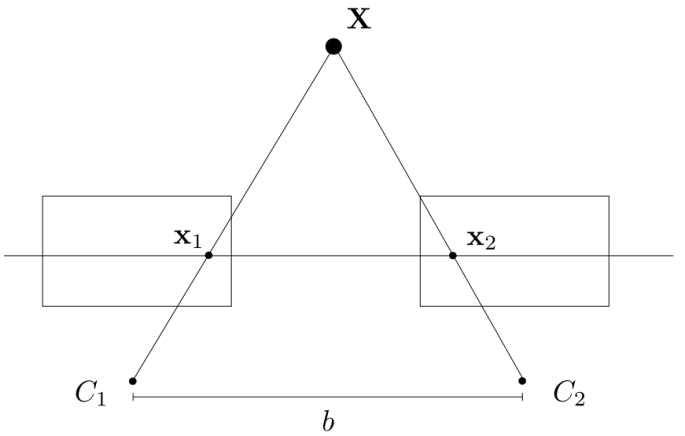

Figure 2.4 An illustration of the stereo view problem inspired by the figure in [Olsson, 2021].

the x-component of the pixel value for both projections is called disparity and is given by

$$d = x_1 - x_2 \quad = \frac{\gamma f b}{X_3} \tag{2.22}$$

The depth of the point can then be recovered from the computed disparity by solving for X_3 in equation 2.22.

Visual features

A common way of finding point correspondences between images are by matching visual features. A visual feature is a local interest point in an image. The location of the feature in an image is often referred to as keypoints. The keypoints have associated descriptors which describes the region in the image surrounding the keypoint by the local pixel gradients. The keypoint descriptors are then used for matching visual features between images [Lowe, 2004].

2.3 Simultaneous localization and mapping

Simultaneous localization and mapping addresses the problem of building a map of the environment, while simultaneously estimating the pose of the robot in said environment. Initially proposed in [Leonard and Durrant-Whyte, 1991] and builds on the idea of the Extended Kalman Filter [Julier and Uhlmann, 2004] where a single state vector is used to represent the estimated pose of the robot and a feature set collected from the environment together with a covariance matrix which is used to represent the associated uncertainty in the pose estimate and features. While the

robot traverses the environment and takes measurements the covariance matrix is updated with the EKF. The state vector grows as new features are observed. The drawback of EKF-SLAM is mainly due to the non-sparsity and quadratic size of the covariance matrix ($(3+2N)\times(3+2N)$, N features, 2D-space) which in turn produces a large computational load as the number of features grow [Thrun and Leonard, 2008].

The probabilistic SLAM formulation

The SLAM problem can be defined in a probabilistic manner [Thrun and Leonard, 2008] [Grisetti et al., 2010]. Let the robot pose at time t be described by $\mathbf{x}_t = (\bar{\mathbf{X}}, \Theta)^T$ where $\bar{\mathbf{X}} = (x, y, z)^T$ is the 3D position of the robot and Θ denotes the orientation. The path of the robot $\mathbf{X}_T = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T\}$ is then characterized by the sequence of poses the robot undertakes from the initial pose $\mathbf{x}_0$ up to time T. The odometry reading $\mathbf{u}_t$ is the relative motion of the robot between time $t-1$ and t. The odometry could for example be given by a motion model where the robot is actuated by a control signal, estimated by visual odometry computed from images, scan matching with a LiDAR or for a wheeled robot, wheel encoder readings.

Let $\mathbf{U}_T = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T\}$ be the sequence of odometry readings up to time T. In theory, the sequence $\mathbf{U}_T$ would be enough to recover the path $\mathbf{X}_T$ from $\mathbf{x}_0$. However, since measurements may be corrupted by noise and the robot may be subjected to external disturbances such as wheel slip, the recovered path may be suffering from significant drift.

Now, let the map of the environment be denoted by $\mathbf{m}$, which contains landmarks or features which could be objects, geometric shapes and $\mathbf{m}$ are their locations. The robot senses the landmarks through measurements $\mathbf{z}_t$. Assuming the robot samples the environment at every timestep t, the sequence of measurements $\mathbf{Z}_T$ up until time T are then given by $\mathbf{Z}_T = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T\}$. The SLAM problem is then the problem of recovering the robot path $\mathbf{X}_T$ from $\mathbf{U}_T$ and $\mathbf{Z}_T$ while simultaneously modelling the map of the environment $\mathbf{m}$ [Thrun and Leonard, 2008]. Figure 2.5 illustrates a graphical model of the SLAM problem.

Online and Offline SLAM The map estimation and the path can be estimated over either the full posterior or up to the current time step. The former is referred to as offline SLAM and the latter is referred to as online SLAM. Online SLAM seeks to recover the present pose $\mathbf{x}_t$ and map $\mathbf{m}$ from measurements and odometry sequences $\mathbf{Z}_T$, $\mathbf{U}_T$ and may be formulated as a probability distribution $p(\mathbf{x}_t, \mathbf{m} \mid \mathbf{Z}_{1:t}, \mathbf{u}_{1:t})$. Offline SLAM seeks to recover the full robot path $\mathbf{X}_T$ and map $\mathbf{m}$ from the full measurement and odometry sequences. The offline SLAM problem may also be formulated as a probability distribution $p(\mathbf{X}_T, \mathbf{m} \mid \mathbf{Z}_T, \mathbf{U}_T)$ [Thrun and Leonard, 2008]. In both instances, the goal is to maximize the probability of the estimated path and map given the measurements and odometry readings.

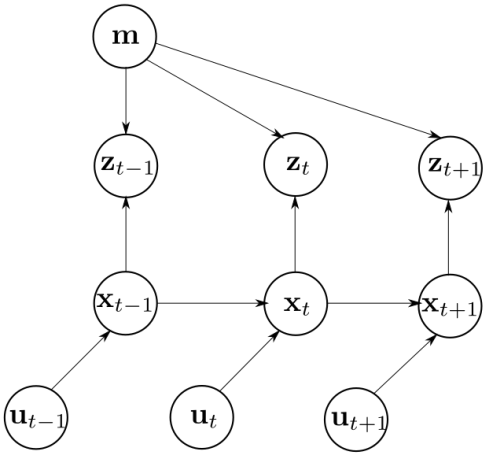

Figure 2.5 A graphical model of the SLAM problem. The drawing is inspired by Figure 37.1 in [Thrun and Leonard, 2008].

Graph based SLAM

In recent years, graph based approaches to the SLAM problem have gained traction as advancements have been made in the field of sparse linear algebra as well as new insights into the structure of the SLAM problem [Grisetti et al., 2010]. The advantage of graph based SLAM over EKF-SLAM is the ability to scale to high-dimensional maps [Thrun and Leonard, 2008].

The graph approach solves the SLAM problem by nonlinear sparse optimization where nodes in the graph correspond to poses and associated measurement of the map at the pose. The edges correspond to spatial constraints produced by odometry readings. If we let the edge constraint be e_{ij} between nodes i and j then the pose graph optimization seeks to minimize the cost function

$$\sum_{e_{ij}} \|\mathbf{C}_i - \mathbf{T}_{e_{ij}} \mathbf{C}_j\|^2 \tag{2.23}$$

where $\mathbf{C}$ represents the camera pose and $\mathbf{T}_{e_{ij}}$ represents the rigid body transformations between the poses [Fraundorfer and Scaramuzza, 2012].

Graph based approaches usually break down the SLAM problem into two separate parts, constructing the graph and optimizing the graph [Grisetti et al., 2010]. The graph construction is usually referred to as the front-end and the graph optimization is usually referred to as the back-end. The front-end is tasked with data association by handling the odometry and measurements of the environment to be added as nodes and edges in the graph. The back-end then solves for the most likely configuration given the soft constraints added by the front-end [Grisetti et al., 2010].

Visual SLAM

Visual SLAM utilizes visual sensors to map and localize within an environment. Visual SLAM relies mainly on initialization, tracking and mapping. During initialization a global coordinate system is defined and initial map of a part of the environment is built in the global coordinate system. During tracking the camera pose is estimated with respect to the map by tracking the reconstructed map in the image. This is done by feature matching between image and map. The camera pose is then estimated from feature correspondences. During the mapping the current map is expanded from 3D reconstruction. To avoid failure of tracking during fast camera motions or other external disturbances relocalization may be used to increase the robustness of the visual SLAM implementation. To correct for drift during mapping where errors in the estimation of camera movement accumulate global map optimization is often used. By rediscovering features in previously visited locations loop constraints may be formed to suppress error in the global map optimization [Taketomi et al., 2017].

Loop closure Visual loop closure algorithms often rely on visual bag of words for forming loop closures [Labbé and Michaud, 2013]. In the visual bag of words approach images are reduced to visual words by the computed image feature descriptors. A vocabulary of visual words can either be pre-trained on *a priori* aquired image data or formed incrementally [Labbé and Michaud, 2013]. For a vocabulary of k words a vector representation $\mathbf{V}_d$ of an image may be formed where $\mathbf{V}_d = (t_1, t_2, \ldots, t_k)$ and t_i is the weighted word frequency computed by the *frequency term – inverse document frequency* [Sivic and Zisserman, 2003]

$$t_i = \frac{n_{id}}{n_d} \log \frac{N}{n_i}. \tag{2.24}$$

Here, n_{id} is the number of occurences of word i in Image d, n_d is the total number of words in Image d, N is the number of images in database, n_i is the number of occurences of word i in the whole database. A loop closure is formed by comparing a query image $\mathbf{V}_q$ to the images in the database. The similarity of two images are determined by the normalized scalar product between the query image $\mathbf{V}_q$ and $\mathbf{V}_d$ [Sivic and Zisserman, 2003].

Metrics for evaluation

To evaluate the performance of SLAM algorithms and visual odometry approaches, the estimated poses are generally compared quantatively to a ground truth measurement. One commonly used metric is the absolute trajectory error (ATE) [Zhang and Scaramuzza, 2018], which computes the root mean squared error along the whole trajectory. For a sequence of ground truth positions $\mathbf{p} = \{\mathbf{p}_i\}, i = 0, 1, \ldots N$, and rotations $\mathbf{R} = \{\mathbf{R}_i\}, i = 0, 1, \ldots N$, and the corresponding estimates $\hat{\mathbf{p}} = \{\hat{\mathbf{p}}_i\}$ and $\hat{\mathbf{R}} = \{\hat{\mathbf{R}}_i\}$, the trajectories are aligned using the initial states by forming the rigid body transformation

$$\begin{aligned} \mathbf{R}' &= \mathbf{R}_0 \hat{\mathbf{R}}_0^T \\ \mathbf{t}' &= \mathbf{p}_0 - \mathbf{R}' \hat{\mathbf{p}}_0 \end{aligned} \tag{2.25}$$

and applying it to the position and rotation estimates to retrieve the aligned position and rotation estimates

$$\begin{aligned} \hat{\mathbf{R}}' &= \mathbf{R}' \hat{\mathbf{R}} \\ \hat{\mathbf{p}}' &= \mathbf{R}' \hat{\mathbf{p}} + \mathbf{t}'. \end{aligned} \tag{2.26}$$

The absolute trajectory error for the position and rotation is defined in [Zhang and Scaramuzza, 2018] as

$$\begin{aligned} ATE_{pos} &= \left(\frac{1}{N} \sum_{i=1}^{N-1} \|\Delta \mathbf{p}_i\|^2\right)^{\frac{1}{2}} \\ ATE_{rot} &= \left(\frac{1}{N} \sum_{i=1}^{N-1} \|\angle(\Delta \mathbf{R}_i)\|^2\right)^{\frac{1}{2}} \end{aligned} \tag{2.27}$$

where the positional error $\Delta \mathbf{p}_i$ and the rotational error $\Delta \mathbf{R}_i$ between ground truth and aligned estimate are given by

$$\begin{aligned} \Delta \mathbf{R}_i &= \mathbf{R}_i (\hat{\mathbf{R}}_i')^T \\ \Delta \mathbf{p}_i &= \mathbf{p}_i - \Delta \mathbf{R}_i \hat{\mathbf{p}}_i' \end{aligned} \tag{2.28}$$

Another metric is the relative pose error (RPE) which instead measures drift over subsets of the trajectory. By dividing the trajectory into sub-trajectories and varying length of the sub-trajectories the pose error can be evaluated over varied distances and as such, allows for additional statistical analysis [Zhang and Scaramuzza, 2018].

There are both advantages and disadvantages to using either metric. The ATE provides a single number for the RMS error of the pose which in turn makes it useful for direct comparison. However, the magnitude of the error depends on where along the trajectory the drift occurs. Deviations at the start of the trajectory will have a greater affect on the magnitude of the error than a deviation near the end of the trajectory [Zhang and Scaramuzza, 2018]. The relative pose error gives additional error metrics and can be used to extract statistical information such as mean and standard deviaton for varying trajectory lengths, but is relatively hard to compute.

2.4 Absolute orientation problem

The absolute orientation problem is the problem of aligning two sets of points where the point sets are acquired in different reference frames [Lourakis, 2016]. For example, points measured in a local reference frame that needs to be mapped into a global reference frame such as a GPS, or the reference frame of a building information model. Given two sets of $N \geq 3$ point correspondances $\{\mathbf{x}_n\}, n = \{1, 2 \ldots, N\}$

measured in a local reference frame, $\{\mathbf{y}_n\}, n = \{1,2,\ldots,N\}$ given in a global reference frame, an affine transformation may be found such that

$$\bar{\mathbf{x}}_n = \lambda \mathbf{R}\mathbf{x}_n + \mathbf{t} \tag{2.29}$$

where $\bar{\mathbf{x}}_n$ are the transformed local points. The transformation is then obtained by minimizing the least squares error

$$\sum_{n=1}^{N} \|\mathbf{y}_n - \bar{\mathbf{x}}_n\|^2 \tag{2.30}$$

The weighted means or centroids of respective clouds are given by

$$\begin{aligned} \mathbf{x}_m &= \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \\ \mathbf{y}_m &= \frac{1}{N}\sum_{n=1}^{N} \mathbf{y}_n \end{aligned} \tag{2.31}$$

It is shown in [Arun et al., 1987] that the solution which minimizes the least squares error in (2.30) is obtained by forming

$$\begin{aligned} \mathbf{q}_n &= \mathbf{x}_n - \mathbf{x}_m \\ \mathbf{q}'_n &= \mathbf{y}_n - \mathbf{y}_m \\ \mathbf{H} &= \sum_{n=1}^{N} \mathbf{q}_n \mathbf{q}_n^{'T} \end{aligned} \tag{2.32}$$

and computing the singular value decomposition of $\mathbf{H}$ where $\mathbf{H} = \mathbf{U}\Sigma\mathbf{V}^T$.

The rotation and translation that minimizes the least squares error in (2.30) is then given by

$$\begin{aligned} \mathbf{R} &= \mathbf{V}\mathbf{U}^T \\ \mathbf{t} &= \mathbf{y}_m - \mathbf{R}\mathbf{x}_m \\ \lambda &= \sqrt{\frac{\sum(\mathbf{y}_n - \mathbf{y}_m)^T(\mathbf{y}_n - \mathbf{y}_m)}{\sum(\mathbf{x}_n - \mathbf{x}_m)^T(\mathbf{x}_n - \mathbf{x}_m)}} \end{aligned} \tag{2.33}$$

Note that $\lambda = 1$ when there is no scale ambiguity, for example when the points are obtained by LiDAR scans.

3

System

This chapter begins by presenting Spot and the available sensor configuration. The chapter then presents the software used and the integration of Spot into the ROS framework. Next, the details regarding the implementation and the packaging of the system into a Docker container for deployment onto the SpotCORE. The chapter is concluded with the results from a working prototype system.

3.1 Spot

Spot is a quadruped robot designed and built by Boston Dynamics. Spot has twelve degrees of freedom (three per leg), weighs 32.5 kg and can reach a max speed of 1.6 m/s. The mobility of Spot makes the robot useful for exploration and inspection of environments where conventional mobile robots with tracks or wheels are not able to traverse. Figure 3.1 displays Spot and the current payload configuration. The base model only has visual perception through the depth cameras mounted on the robot body. The payloads consist of a Velodyne VLP-16 LiDAR, a Spot CAM pan-tilt-zoom camera and the SpotCORE, a small form factor PC for onboard computing. Spot can be controlled with the provided tablet and through teleoperation. However, Spot is not bound to operator input and can navigate autonomously by setting up autowalk missions with the tablet.

Cameras

Spot is equipped with five built-in depth cameras. The cameras on Spot are grey scale IR depth cameras and offer reliable depth sensing up to four meters. The IR projection allows the cameras to estimate depth in feature sparse environments. Figure 3.2 shows a stitched view of all five depth cameras and corresponding depth map. The brightness and contrast of the depth map has been increased for the purpose of visualization. Figures 3.3, 3.4 and 3.5 show the two front facing cameras, back facing camera and right facing camera, respectively. The left facing camera has been omitted but is mounted in a mirrored configuration to the right facing camera.

Figure 3.1 Spot with the current payload configuration.

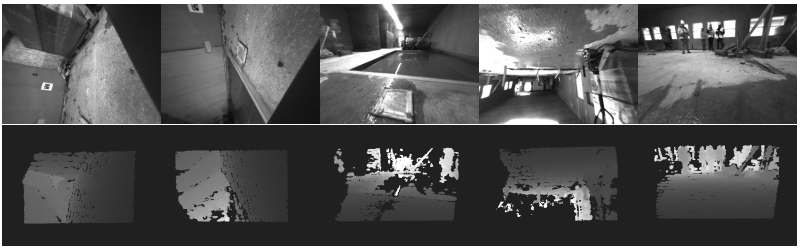

Figure 3.2 Stitched grey-scale image and corresponding depth map.

Table 3.1 Camera extrinsics for each of the depth cameras w.r.t the body frame.

Camera	$x[m]$	$y[m]$	$z[m]$	q_0	q_1	q_2	q_3
Front left	0.3839	0.0318	-0.0423	0.1417	0.8096	0.2216	0.5247
Front right	0.3869	-0.0422	-0.0441	-0.1465	0.8074	0.2296	0.5234
Left	-0.0895	0.1114	0.0405	-0.8007	-0.0034	0.0005	0.5990
Right	-0.0919	-0.1106	0.0335	0.7912	0.0107	0.0003	0.6114
Back	-0.4196	0.0362	0.0131	0.5645	0.5621	-0.4307	-0.4241

Camera extrinsics When quering an image sensor on Spot, the image response is accompanied by the homogenous transformation from body frame to image sensor frame. Table 3.1 lists the camera extrinsics relating the image frame to the body frame. The translation is given in meters. The rotation is given in unit quaternions (q_0, q_1, q_2, q_3). The extrinsics have been obtained by querying each sensor. Since the depth image is registered to the image frame, the same transformation is used.

Figure 3.3 Front cameras.

Figure 3.4 Back camera.

Figure 3.5 Right camera.

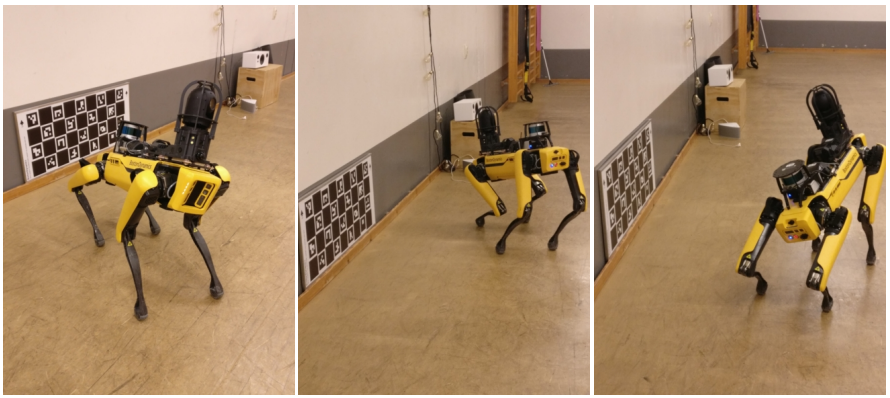

Figure 3.6 An image collage illustrating Spot's posing during the calibration routine.

Camera intrinsics Spot has its own automatic calibration procedure for estimating the camera intrinsics. A board with charuco markers[1] is used, placed horizontally. During the calibration Spot positions itself accordingly see Figure 3.6.The camera intrinsics are stored on Spot. The camera intrinsics for a given sensor may then be extracted from the data received by querying the image sensor.

Payload configuration

As mentioned previously, Spot is equipped with the SpotCORE and a LiDAR, see Figure 3.7. The SpotCORE and LiDAR are bundled into what is referred to as the Enhanced Autonomy Package (EAP). The LiDAR extends Spot's ability to local-

[1] https://docs.opencv.org/4.5.5/df/d4a/tutorial_charuco_detection.html Accessed:2022-02-27

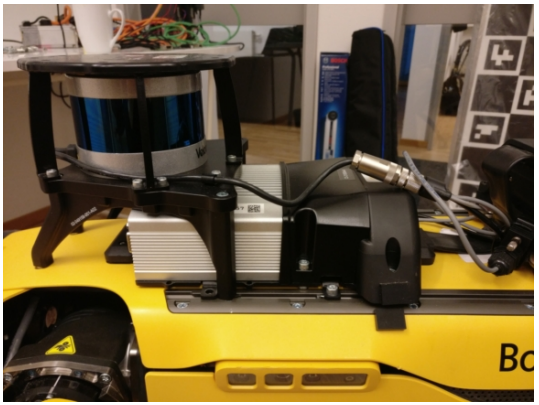

Figure 3.7 The SpotCORE and VLP-16 LiDAR.

ize during autonomous missions in feature sparse environments where the cameras provide insufficient information.

LiDAR The LiDAR sensor is a Velodyne VLP-16. The VLP-16 has a range of up to 100 meters, a 30 degree vertical field of view and measures roughly 300 000 points per seconds. The measurements have a range accuracy of ± 3 cm.

SpotCORE The SpotCORE is a small form factor PC and comes preconfigured with Ubuntu 18.04. The hardware includes an 8th generation Intel i5 processor, 16 GB of RAM and 512 GB of SSD storage.

Frames

During operation certain frames[2] are known to Spot. The frames can be sectioned into inertial frames, robot frames and sensor frames. An inertial frame is a static frame. Two inertial frames are available, *odom* and *vision*. Both inertial frames are defined at the starting location of Spot. However, the intertial frames do not necessarily coincide. Issues related to the placement of the intertial frames and the solution are further elaborated on in Section 3.3. The robot frames consist of the body frame and actuator joints. The body frame is placed in the middle of the robot, between the hip joints. The joint positions are given in the body frame. The sensor frames relate the optical centers of the cameras and velodyne position to the body frame.

The position of the body frame in either intertial frame is continously estimated during operation. The estimated position of the body frame in the odom frame is purely based on the kinematic model of the robot. The estimated position of the body frame in the vision frame is based on the kinematics and an internal visual odometry approach using the depth cameras.

[2] https://dev.bostondynamics.com/docs/concepts/geometry_and_frames Accessed:2022-03-26

Figure 3.8 One of the two DB12 ports on Spot which provide the payloads with power and access to the internal network.

Networking

Communicating with and operating Spot requires a network connection to the robot. Spot hosts its own IP network. The network may be accessed either through a direct connection with an RJ45 cable or by connecting to the wireless network provided by the onboard WiFi access point. The onboard payloads have access to the network through the payload ports which also provide power.

3.2 Software

This section aims to preface the implementation by describing the software that has been used in this thesis and the motivation behind the usage. The following subsections go into detail regarding the Spot SDK, ROS, RTABMap as the selected SLAM solution and finally the usage of docker for deployment of the written software.

SDK

Interfacing with and programming Spot is done through the Spot SDK [3]. The SDK is available as a Python API, and this implementation relies on SDK version 2.3.6 and Python 3.7. The SDK provides extensive functionality and is divided into three types of services: *core*, *robot* and *autonomy*. The core part of the SDK handles low level services such as authentication for acquiring access to higher level services, time synchronization for correcting time drift between the internal clock and an application clock, and a lease for acquiring motor control. The robot service handles access to the robot state, sensors, and robot command. The autonomy services give the user high level control of the robot for setting up missions and autonomous navigation through GraphNav. The services are accessed through a service-client

[3] https://dev.bostondynamics.com/docs/concepts/readme Accessed:2022-01-02

model, where applications that want to access the services register a corresponding client.

The extent of which the SDK is used in this thesis is limited to the core and robot services for authentication and extracting odometry, RGB-D images, and point cloud data. The functionality of the autonomy services have thus not been explored.

Robot Operating System

The Robot Operating System [Quigley et al., 2009], abbreviated *ROS*, is despite its name a software framework rather than an operating system. ROS is designed to ease process communication as well as cross-system communication. The communication is peer-2-peer based. The processes referred to as nodes, registers with a ROSmaster service which handles the look-up. Nodes are typically simpler programs that handle single tasks. Information shared between nodes are sent and received as messages. Messages are transmitted and received using message topics which the nodes can either subscribe or publish to.

One of the advantages of ROS is the generalized sensor message format used. Having the sensor data adhere to a generalized message format makes the integration of a sensor into a system rather trivial.

RTABMap

The implementation in this thesis relies (solely) on RTABMap as the SLAM library for building dense reconstructions. Since it was not clear at the start of the thesis whether or not the payload configuration would be available or to what extent, the focus was directed towards existing visual SLAM solutions where the depth cameras on Spot could be integrated. The visual SLAM package ORB-SLAM [Campos et al., 2021] was considered a candidate early on in the thesis since it supported RGB-D cameras and had ROS support. Unfortunately, ORB-SLAM only outputs sparse feature maps, which made it unsuitable for this task. Ultimately the decision was made to use RTABMap due to its many features, including ROS-support. LiDAR-only SLAM solutions were not considered at all due to the previously mentioned reasons.

RTABMap was originally proposed as an apperance-based loop-closure solution [Labbé and Michaud, 2013] but has developed into a full online graph-based visual SLAM solution with multi-camera support and dense scene reconstruction [Labbé and Michaud, 2019]. RTABMap only works with cameras that have depth information, such as stereo or RGB-D image sensors. Although mainly a visual SLAM solution, RTABMap can also utilize 2D laser scans and 3D LiDAR pointclouds for ICP odometry, link refinement in the graph optimization and for generating occupancy grid maps. The LiDAR pointclouds can also be used for generating dense reconstructions. RTABMap can be used with external odometry instead of the included visual odometry. The external odometry may for instance be generated from

a motion model of the robot or from a separate visual odometry approach. When external odometry is used, the visual information provided is only used for finding loop closures and during map reconstruction. The loop closure algorithm uses the visual bag of words approach [Labbé and Michaud, 2013]. However, the vocabulary is not pretrained but built during operation. RTABMap is available as both a C++ library and as a ROS package.

Docker

Docker is a solution for packaging software into smaller, isolated environments called containers. Similiar to virtual machines, a Docker container is a system environment that is isolated from the host system, but can allocate resources from the host system. The full environment can be packed into a Docker image which can then be deployed onto any host system that also runs Docker. The image contains a file system with all dependencies and configurations that is needed to run the software. For ease of development and testing, Docker was chosen early on for deploying the software onto the SpotCORE since the development, testing and building the environment could be done on a separate machine.

3.3 Implementation

This section describes the implementation and integration of Spot's sensors into ROS and packaging of the software into a Docker container for deployment onto the SpotCORE. First, the architecture and design choices regarding the implementation are described. Secondly, the docker deployment is briefly described. Finally, the initial impressions from a working system mapping an office space are presented.

Architecture

The architecture is built upon ROS for data acquisition and intraprocess communication with the ROS implementation of RTABMap. The architecture can be divided into two different pipelines. Acquiring sensor data and robot state information will be referred to as the sensor pipeline and the mapping process will be referred to as the SLAM pipeline.

Sensor pipeline The sensor pipeline wraps the SDK calls and converts the acquired sensor data into ROS messages. The ROS messages are then published to the respective topics. A ROS node exists for each "sensor" type for fetching RGB-D data, LiDAR pointclouds and odometry. At launch the ROS nodes follow the same procedure and can be summarized in the following steps:

- Read ROS parameters
- Authenticate

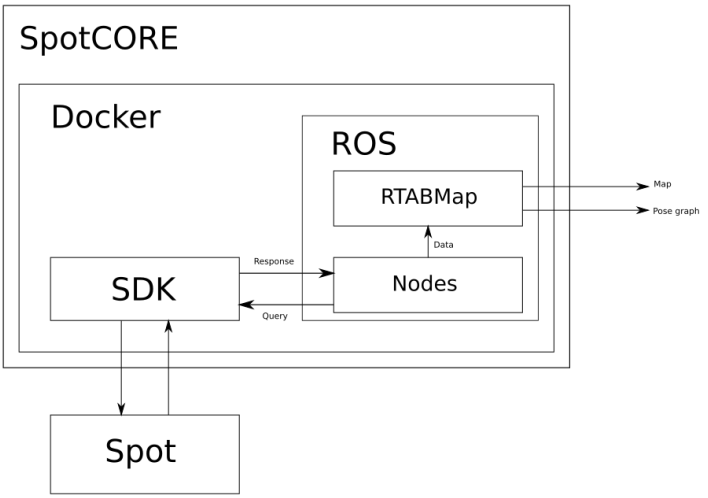

Figure 3.9 An illustration of the deployed system.

- Initialize ROS node
- Publish data

At the intial step the ROS parameters are read from a launch file which specifies the robot IP address and credentials. For the camera node, the positioning of the camera, e.g., "frontleft" is also specified. At the authentication step, a SDK object is created which specifies the client to be registered, and the credentials for accessing the robot is passed through. During the initialization step, the ROS node is registered and the publishers are created. In the last step of the procedure the program enters the loop in which the calls to the SDK are made for acquiring data. The data is then handled and packaged into ROS messages and published.

SLAM pipeline The SLAM pipeline handles the configuration of RTABMap and the sensor topics. An illustration of the SLAM-pipeline is shown in Figure 3.10. The pipeline is additive in the sense that any amount of cameras may be used. For each camera, the output topics are synchronized into *RGBDImage* messages using the RTABMap *rgbd_sync nodelet* which packages the RGB, depth and camera info into a single message. This is done to allow RTABMap to subscribe to multiple depth camera topics. RTABMap also subscribes to the odometry provided by the odometry node and the pointcloud from the LiDAR. The pipeline utilizes the

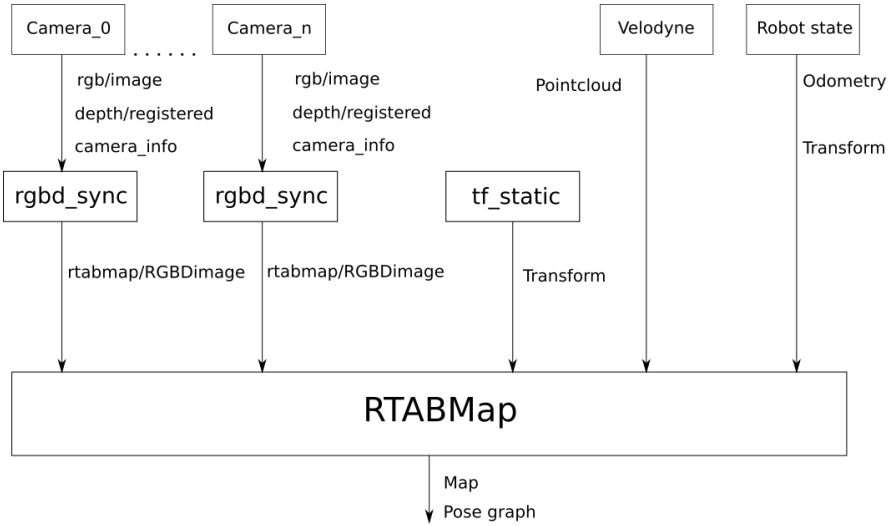

Figure 3.10 An illustration of the SLAM pipeline.

tf_static package which publishes the static transformations from the body frame to the camera optical frames. During the mapping RTABMap outputs the pose graph estimate and the current map. The default update rate of the pose graph is set to 1 Hz and determines the rate of which the keyframes are read. Faster update rates allow for more dynamic behavior and finding more loop closures. However, the computing power available constrains the update frequency and as such, sets an upper bound on the update rate under which realtime issues are avoided. Realtime issues occur when the time it takes to update the pose graph exceeds the set update rate of the graph.

Nodes

For every type of "sensor" in the sensor pipeline, a ROS node has been written. Specifically, a camera node which handles the depth cameras, a velodyne node which publishes the LiDAR point clouds, and an odometry node which publishes the robot odometry and transformations.

Camera The camera node registers an image client service from which RGB and depth images are acquired. When registering the image client, the camera position is specified. Additional options exist for fetching rectified images and depth images which are pre-registered to the RGB frame. This eliminates the need for using additional rectification and depth registration pipelines, although ready ROS solutions exist such as *image_proc*[4]. The image data consists of *numpy* arrays and are cast

[4] http://wiki.ros.org/image_proc Accessed:2022-01-02

into the OpenCV format. The image data can then be directly converted into ROS image messages using the ROS-OpenCV bridge.

As mentioned in Section 3.1 the camera intrinsics have been retrieved in advance and stored in a YAML file. When the node is launched, the YAML is loaded and read and the contents are written to a *CameraInfo* message. Since the depth image is registered to the RGB frame, the same camera intrinsics are used as for the RGB image.

The camera node is set to publish at a rate of 30 Hz, which the system seems to handle when using a single camera. However, using all five cameras the data transmission seems to saturate the network and the increased bandwidth usage lowers the received message rate to roughly 17 Hz.

Velodyne The Velodyne node registers a *velodyne_pointcloud_client* to poll for pointcloud data. The pointcloud data is recieved as *numpy* array, and are written to a *PointCloud2* message. The frame of which the retrieved pointcloud were registered in caused a great deal of confusion during this part of the implementation. After some investigation it turned out that the pointcloud were given in the inertial *odom* frame, and not the frame of the velodyne. The documentation does not specify the reason behind the frame usage. One explanation could be that Spot uses the LiDAR for localization when visual odometry can not be computed. It could be that if the visual odometry is unavailable the robot tries to estimate its movement based on the kinematics and ICP odometry and as such, uses the *odom* frame.

Odometry The odometry node publishes *odometry* messages which maps the body frame in the vision frame. The odometry node registers a robot state client. The client returns robot state messages where the robot pose, velocities and frame transformations can be extracted. The information acquired is packaged into a *Nav_msg /Odometry* message and published.

As previously mentioned, the pointcloud from the velodyne is registered in the odom inertial frame. For RTABMap to use the pointcloud, the transformation from the odom frame to the vision frame need to be known. Since the transformations from both inertial frames to the body frame are available in the *robot state* message, the transformation from the vision frame to the odom frame can simply be found by computing the transformation

$$\mathbf{H}_{odom}^{vision} = \mathbf{H}_{body}^{vision} (\mathbf{H}_{body}^{odom})^{-1} \tag{3.1}$$

Finally, $\mathbf{H}_{odom}^{vision}$ is published as a *tf* message.

Software deployment

The software written in this thesis has been packaged into a Docker image. The Docker image is built using the Ubuntu 20.04 base image. The image contains the RTABMap binaries, a ROS Noetic workspace environment with the launch files, nodes and a Python3.7 virtual environment in which the SDK is

installed. Since the RTABMap binary available in the ROS repository does not have multi-camera support RTABMap had to be built from source with the flag **-DRTABMAP_SYNC_MULTI_RGBD=ON** .

The software is deployed onto the SpotCORE by exporting the image as a tar file and transferred onto the SpotCORE. The image is then loaded by importing the tar file. When running the container certain permission flags have to be set, specifically the

- **–net==host** flag which exposes the host network to the container.
- **–env="DISPLAY"** which allows the container to use the XServer of the host system.
- **–volume="$HOME/.Xauthority:/root/.Xauthority:rw"** which mounts the host Xauthority in the container.
- **-v <host folder>:/path/to/container/folder** which mounts a folder on the host file system into the container for shared access.

3.4 Initial test and impressions

After having a working system for data collection and mapping a test run was performed. The robot was operated manually with the tablet during which the odometry, camera data and LiDAR pointclouds were recorded to a *rosbag*. After the run, the data was replayed from the *rosbag* to RTABMap. After the mapping procedure, RTABMap saves the keyframes, pose graph and odometry estimates and reconstruction to a database file. The database file may then be read by the *rtabmap-databaseViewer* tool. From here, the odometry and pose graph estimates and camera and LiDAR reconstruction were exported.

Figures 3.11, 3.13 and 3.15 illustrate the reconstruction from the depth cameras. The point cloud has been downsampled a factor of 4 and the max depth has been set to 1.5 m. The map created from the depth cameras is deemed inadequate. Due to the orientation of the cameras where they all face slightly downwards, surfaces that are close to and higher than Spot are hard to capture. The depth information suffers from increased noise for longer distances. However, the geometry that is captured seems correct. For instance, points that should lie on a plane do seem to lie on a plane, such as the ground, walls and other plane surfaces.

Figures 3.12, 3.14 and 3.16 illustrate the LiDAR reconstruction from the LiDAR pointclouds captured during the same test run. The pointcloud has been downsampled a factor of 2 and the cloud has been manually segmented using CloudCompare[5] for illustration purposes. The reconstruction seems very accurate. This did not come

[5] https://www.danielgm.net/cc/ Accessed: 2022-02-25

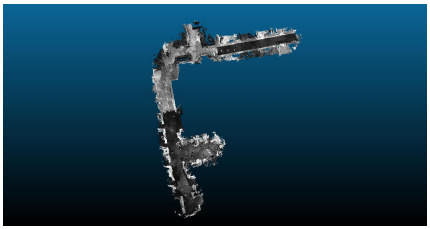

Figure 3.11 A top view of the camera reconstruction.

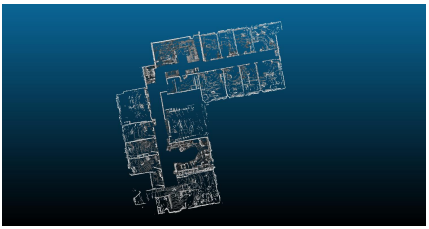

Figure 3.12 A top view of the LiDAR reconstruction.

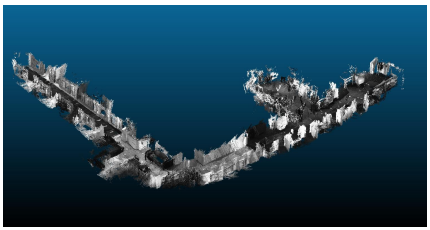

Figure 3.13 A side view of the camera reconstruction.

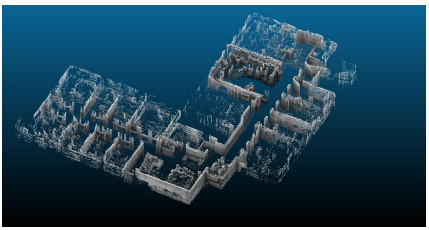

Figure 3.14 A side view of the LiDAR reconstruction.

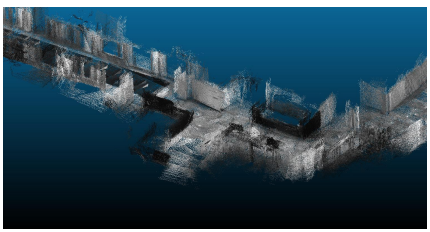

Figure 3.15 Close up view of dining area in the camera reconstruction.

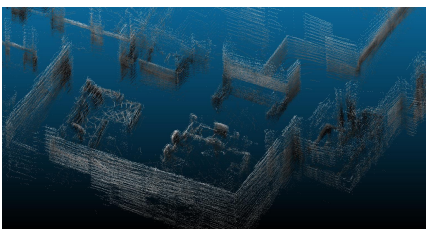

Figure 3.16 Close up view of dining area in the LiDAR reconstruction.

as a great suprise given the range accuracy of the LiDAR. However, the clouds are still captured from a moving platform which could introduce slight misalignments.

Figure 3.17 shows the robot path in the X-Y plane for the odometry and pose graph output. Without any actual ground truth for the path, it seems reasonable compared to what was observed visually during the test run and when considering the layout of the space. Slight misalignment between the odometry and the pose graph can be deduced from the figure and the misalignment seem to be at its largest when the robot made sharp turns, such as in the top half of the figure.

The conclusion that was drawn from the initial test was that the combination of LiDAR and visual information would be the way forward. As mentioned, the cam-

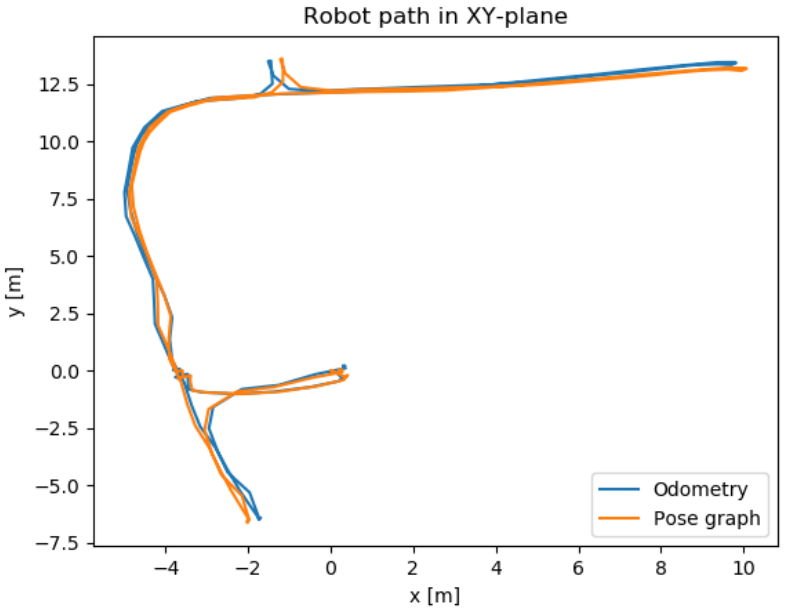

Figure 3.17 The robot path in the X-Y plane estimated by the odometry and pose graph during the test run.

era reconstruction was deemed inadequate for any real usage in the comparison to the building information model or for acquiring any meaningful geometric information other than the layout of the space visited. However, the visual information is still useful for localization purposes and for finding loop closures to correct for the odometry drift. The pointcloud captured by the LiDAR provides seemingly accurate reconstructions. The only drawback is that the pointclouds are untextured and can be hard to interpret without some additional manual intervention.

4

Results

This chapter presents the results from the experiments conducted to benchmark the localization performance of the system and evaluating the accuracy of the reconstructions. The results are two-part. Section 4.1 describes the localization experiments conducted at Brunnshögs stationstorg where positional ground truth data were collected using GPS tracking. Section 4.2 describes the evaluation of the reconstructions produced from data collected at the construction project Vipan.

4.1 Localization

This section describes the experiments conducted to evaluate the localization performance of RTABMap when utilizing the odometry output from the robot. The experiments were conducted at Brunnshögs stationstorg, Lund. The location was chosen to minimize interference from reflections on the GPS signal since the surrounding environment is flat and void of tall structures. See Figure 4.1 for an image of the location.

Experimental setup

The robot was equipped with two realtime-kinematic global positioning systems (RTK-GPS), see Figure 4.2. The RTK system can track the position of an object in realtime. The measurements are then corrected against a fixed station which is located in the vincinity. The fixed station continously sends GNSS information to Swepos [1]. Under the right conditions, the measurements from the RTK-GPS system may acquire a precision in the range of centimeters [2].

The intention was to track the robot's position using the GPS and estimate the heading of the robot by computing the yaw from the two measured GPS signals. The measured pose would then serve as ground truth for the experiment from which the

[1] https://www.lantmateriet.se/en/maps-and-geographic-information/gps-geodesi-och-swepos/swepos/om-swepos/ Accessed: 2021-01-10

[2] https://www.lantmateriet.se/en/maps-and-geographic-information/gps-geodesi-och-swepos/GPS-och-satellitpositionering/Metoder-for-GNSS-matning/RTK/ Accessed: 2022-01-10

Figure 4.1 An image of the location in Brunnshög where the localization experiments were conducted.

ATE and RPE could be computed. To aid RTABMap in finding loop closures, three fiducials were taped to separate wooden boards and placed around the starting locations of the experiments. See Figure 4.3 for the placement of the fiducials. Multiple experiments were performed. The robot was operated manually with the tablet during which the GPS data, odometry and sensor data were recorded. The paths taken during each experiment started and ended in the same location.

Clarifications regarding the experimental data

Accurate and reliable ground truth measurements are not always easy to acquire. During the experiments at Brunnshög the difficulties related to tracking a moving object in realtime with global positioning made themselves evident. First off, the GPS measurements from the RTK system placed at the front could not acquire a high enough resolution which resulted in frequent jumps in the positioning. The placement of the antenna close to the Spot CAM seem to have caused additional reflections and as such rendered the measurements unusable. The GPS receiver mounted on the back of the robot was afflicted by the same jumps in the positioning during parts of the trajectories. Due to time constraints the experiments could not be repeated or refined within the scope of this thesis. Instead of discarding all the results, the descision was made to evaluate the performance of the localization on the available data. Since the measurements from the GPS system placed at the front

Figure 4.2 The two GPS systems mounted on Spot.

Figure 4.3 Placement of the fiducials during the localization experiments.

could not be included the heading had to be estimated using only the signal, i.e, the GPS system placed at the back. Therefore, only selected parts of the robot path where the ground truth measurement was smooth were used in the evaluation.

Method

To compare the estimated pose with the ground truth measurement, the estimated poses and ground truth had to be registered to a shared coordinate frame. The odometry, pose graph and GPS signal have been aligned by manually picking point correspondances and solving the absolute orientation problem, as presented in Section 2.4. See Appendix A.1 for the python implementation. The resulting alignment for one of the experiments are shown in Figure 4.4 in the x-y plane and in the x-z plane in Figure 4.5. The figures make the previously mentioned issues related to the GPS

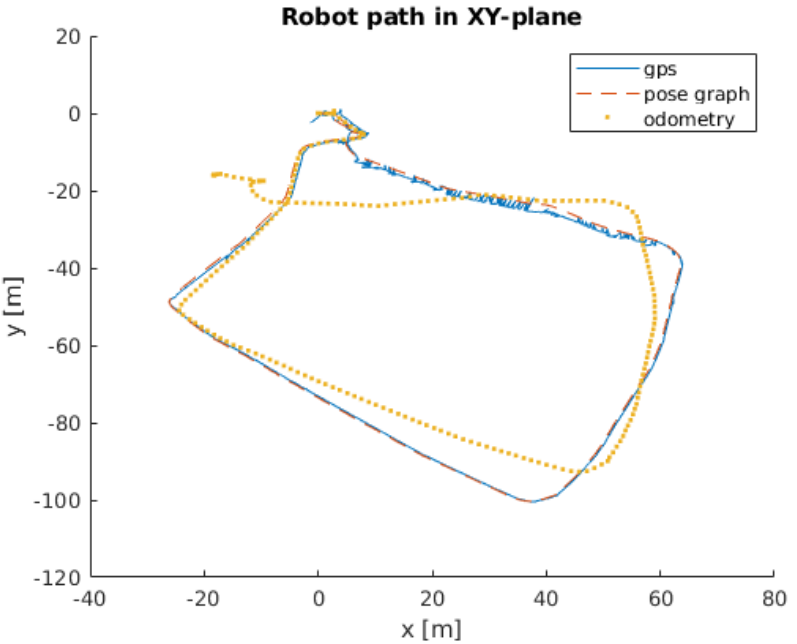

Figure 4.4 The robot path in the XY-plane measured by GPS and estimated by odometry and pose graph.

tracking clear. However, notice that the jumps in the positioning is mainly affecting the position on the z-axis. Also, note the scale of the axes.

Since the odometry and pose graph estimates are given in current ROS time and the GPS measurements in world clock the data had to be manually aligned in time. Figure 4.6 illustrates the aligned x-y-z trajectories for the same experiment.

The yaw angle is estimated by computing the difference in measured x and y position between samples. The yaw angle θ_n at sample n is given by

$$\theta_n = \arctan\left(\frac{y_{n+1} - y_n}{x_{n+1} - x_n}\right). \tag{4.1}$$

The time- and position-aligned data is then written to a file to be used with the evaluation methods presented in Section 2.3 and described in the paper by [Zhang and Scaramuzza, 2018]. The paper is accompanied by a toolbox written in Python and is available at github[3]. The toolbox provides functionality for selecting parts of the trajectories to be used in the evaluation and alignment of the ground truth

[3] https://github.com/uzh-rpg/rpg_trajectory_evaluation Accessed: 2022-01-02

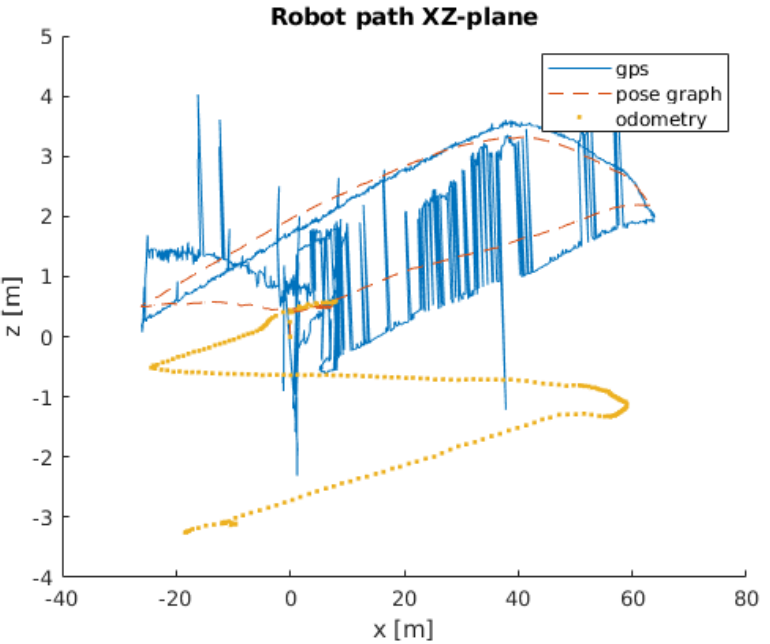

Figure 4.5 The robot path in the XZ-plane measured by GPS and estimated by odometry and pose graph.

and estimate. The estimated yaw angle has been used to fine tune the alignment of the trajectories using the SE3 setting which computes a translation and a rotation around the z-axis.

Performance

The results from three experiments are presented here. As previously mentioned, only parts of the experiments are presented due to the issues related to the GPS measurements. The error metrics for the orientation have been omitted in the presentation of these results. This is motivated by the uncertainty of the ground truth measurements since the heading had to be estimated from a single GPS signal.

Figures 4.7, 4.8, 4.11, 4.12, 4.15, and 4.16 illustrate the aligned paths in the XY plane for the ground truth and pose graph and ground truth, and odometry, respectively. The odometry is Spot's own pose estimation based on the kinematics and visual information from the depth cameras. The pose graph is the estimated pose that RTABMap outputs with the odometry from Spot and the depth cameras as input.

The first experiment ran on a stone and concrete path with a slight slope. The

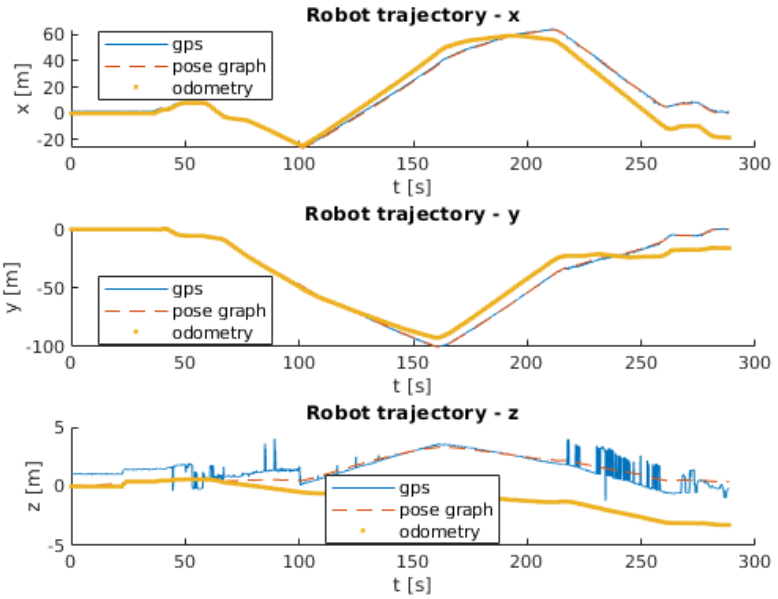

Figure 4.6 The robot trajectory measured by GPS and estimated by odometry and pose graph.

path included multiple turns and made circular patterns. The pose graph deviated slightly more from the ground truth towards the end of the path. The deviations are due to no global loop closure being found, meaning no constraint was added between the starting and ending location. However, the circular path taken resulted in multiple local loop closures which corrected for the drift along the path.

The latter two experiments followed roughly the same path, in the shape of a square along the paved road. Global loop closures between starting and ending location were found during both experiments. It is interesting to note the significant shift in performance between the second and third experiments since the conditions for the experiments were the same. No other explanation to the worsened performance besides the increased odometry drift in the third experiment has been found.

Figures 4.9, 4.10, 4.13, 4.14, 4.17 and 4.18 illustrate the position drift between ground truth and pose graph and ground truth and odometry in x-y-z, respectively.

Tables 4.1 and 4.2 present the absolute trajectory error statistics for the pose graph and odometry, respectively. It is clear from comparing the tables that the pose graph reduces the absolute trajectory error for all experiments. Sub-meter accuracy is acheived for paths longer than 100 meters. Similar ATE is acheived for two dif-

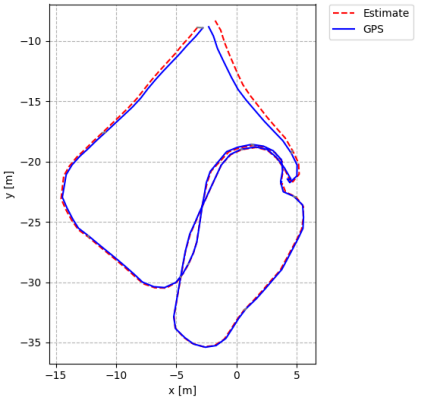

Figure 4.7 Path of the robot in the XY-plane measured by GPS and estimated by pose graph during the first experiment.

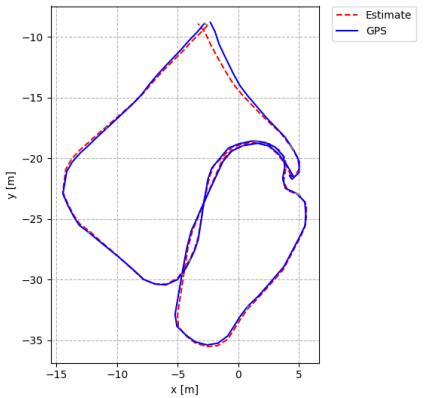

Figure 4.8 Path of the robot in the XY-plane measured by GPS and estimated by odometry during the first experiment.

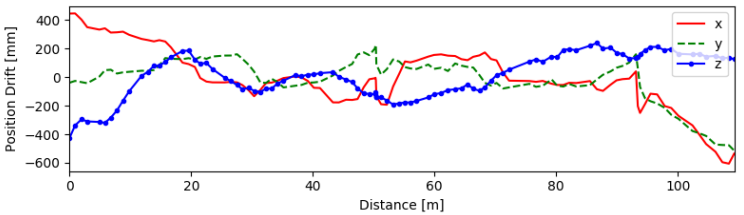

Figure 4.9 The position drift between GPS and pose graph during the first experiment.

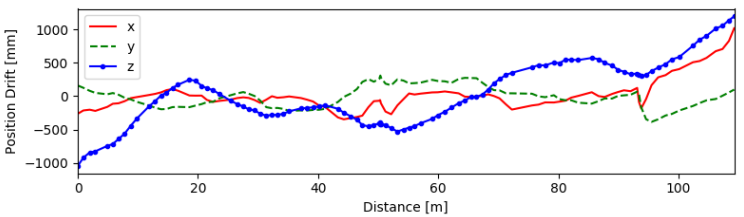

Figure 4.10 The position drift between GPS and odometry during the first experiment.

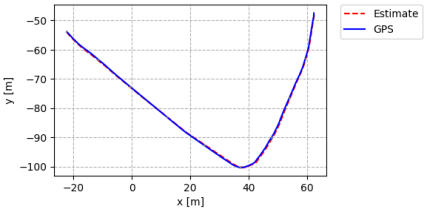

Figure 4.11 Path of the robot in the XY-plane measured by GPS and estimated by pose graph during the second experiment.

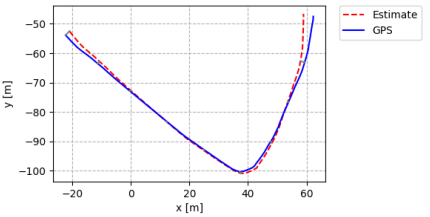

Figure 4.12 Path of the robot in the XY-plane measured by GPS and estimated by odometry during the second experiment.

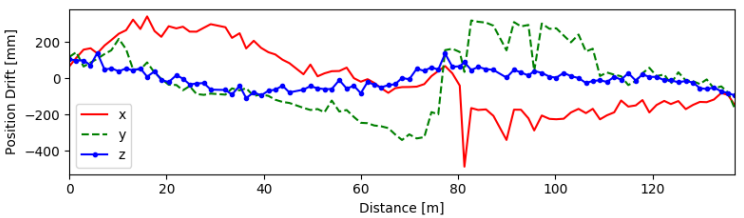

Figure 4.13 The position drift between GPS and pose graph during the second experiment.

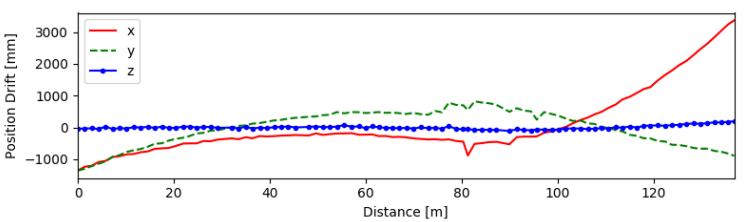

Figure 4.14 The position drift between GPS and odometry during the second experiment.

ferent scenarios where the first experiment found multiple local loop closures and the second experiment had no local loop closures but instead a global loop closure at the end of the path. See Appendix A.2 for the resulting reconstructions from the experiments.

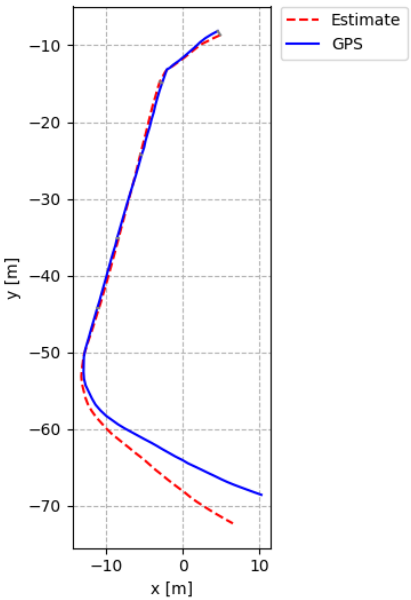

Figure 4.15 Path of the robot in the XY-plane measured by GPS and estimated by pose graph during the third experiment.

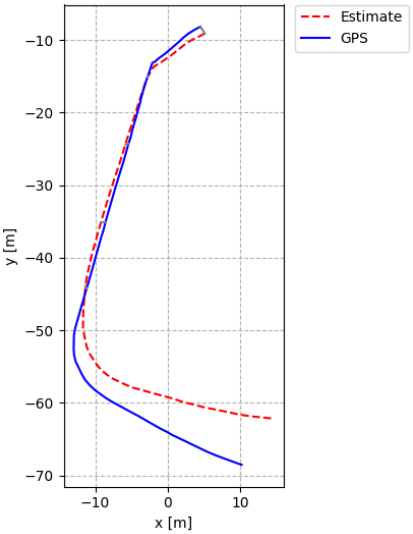

Figure 4.16 Path of the robot in the XY-plane measured by GPS and estimated by odometry during the third experiment.

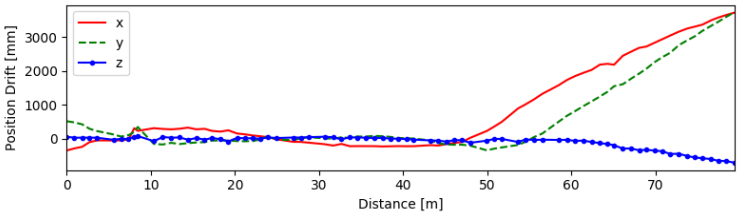

Figure 4.17 The position drift between GPS and pose graph during the third experiment.

4.2 Vipan

Vipan is an ongoing construction project in south-eastern Lund. The ground floor of the building has been used as a test bed for this project. Figure 4.19 shows the model of the planned building. The available building information model, abbreviated *BIM*, is used for comparison when evaluating the accuracy of the reconstruc-

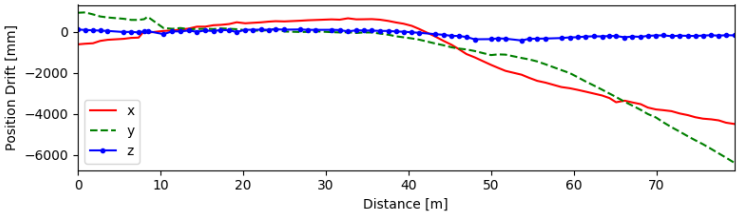

Figure 4.18 The position drift between GPS and odometry during the third experiment.

Table 4.1 The absolute trajectory error of the pose graph estimate for the translational part for Experiments 1-3.

Run	**RMSE** [m]	**Std.dev** [m]	**Mean** [m]	**Max** [m]	**Min** [m]
1	0.3018	0.1593	0.2564	0.7834	0.0497
2	0.2579	0.0783	0.2457	0.4982	0.1172
3	2.0282	1.5864	1.5864	5.308	0.0428

Table 4.2 The absolute trajectory error of the odometry estimate for the translational part for Experiments 1-3.

Run	**RMSE** [m]	**Std.dev** [m]	**Mean** [m]	**Max** [m]	**Min** [m]
1	0.5475	0.2782	0.4716	1.5709	0.0869
2	1.1837	0.7302	0.9317	3.4865	0.2475
3	3.2131	2.3062	2.2373	7.8008	0.1708

tion. For this purpose, a pointcloud has been sampled from the BIM. The points are sampled on every surface of the ground floor at a density of 20 points per square meter. Figures 4.20 illustrate two views of the pointcloud sampled from the BIM.

The data used in the reconstruction has been collected during a mission created using the autowalk function available in the tablet. Spot navigates autonomously between predefined positions in the building. During the mission the odometry, RGB-D images, and LiDAR pointclouds are recorded to a *rosbag*.

Reconstruction

Figure 4.21 illustrates the resulting LiDAR reconstruction from a recorded autonomous mission. The mapping is done in "pseudo realtime" where the input data to the SLAM pipeline is played back from the *rosbag*. After the mapping, the map, keyframes and odometry are saved to a database file and read with the provided rtabmap-databaseViewer tool. The pointcloud is then exported to a *.pcd* file. The

Figure 4.19 A front view of the building information model of Vipan.

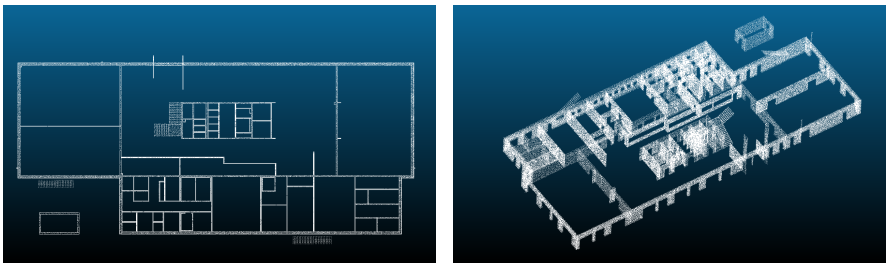

Figure 4.20 A top view and side view of the sampled BIM.

pointcloud has been post-processed using the free software CloudCompare[4] where the cloud has been manually segmented to remove certain parts of the reconstruction such as the floor and ceiling to ease visualization. Light rays are then projected along the z-axis using the ambient occlusion tool which results in a color gradient in the cloud, also for the purpose of visualization.

Aligning reconstruction with model

To get an overall impression of the resulting reconstruction from the mapping, the reconstruction was aligned to the pointcloud sampled from the BIM. Figure 4.22 illustrates the reconstruction aligned to the pointcloud sampled from the BIM. Here, the pointcloud from the reconstruction is first roughly aligned by picking a number of point correspondences with the *point selection tool* in CloudCompare. The resulting rotation and translation are then used as an initial guess to the *fine registration tool* in CloudCompare which uses iterative closest point (**ICP**) for additional alignment of the pointclouds.

[4] https://www.danielgm.net/cc/ Accessed: 2022-02-25

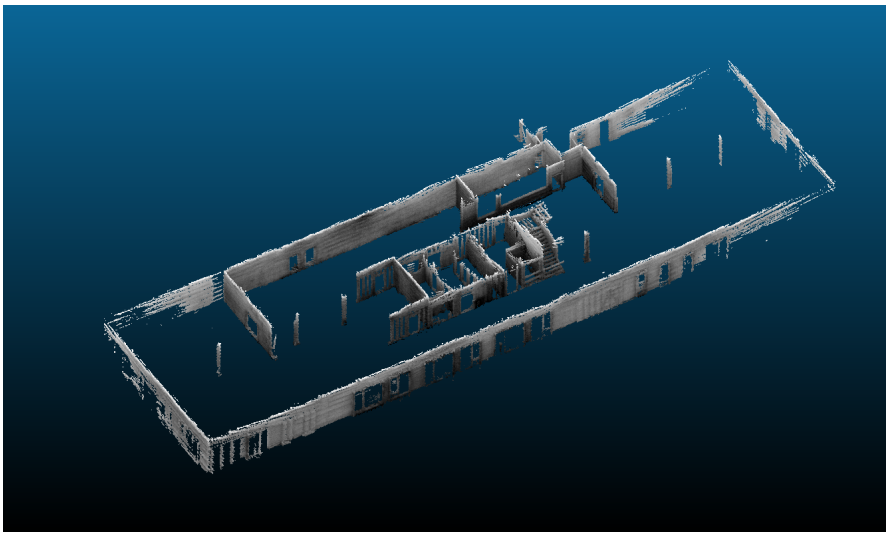

Figure 4.21 The reconstruction visualized in CloudCompare.

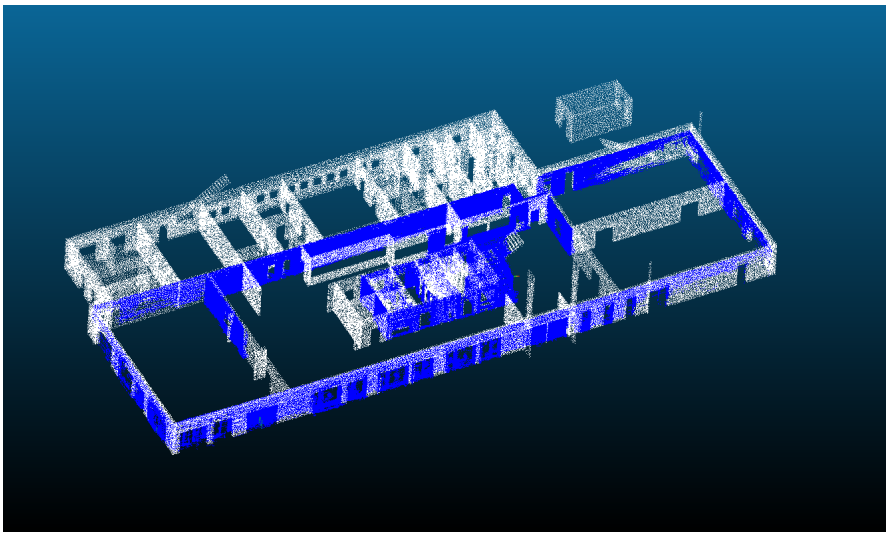

Figure 4.22 Reconstruction (blue) aligned with BIM pointcloud (white).

Accuracy

To answer the question: "How accurate is the reconstruction?", two very rudimentary approaches were used in the evaluation. First, distances between known geometry in the reconstruction are compared to the same geometry in the BIM. Secondly, the resulting LiDAR cloud registration from the mapping is evaluated. This is done by selecting pieces of the reconstruction where the geometry of the scene is known. A geometric feature is fitted to parts of the reconstruction. The distance between the points that make up the geometry and the feature model is then computed.

Distances The distances in the reconstruction and the BIM are measured in CloudCompare using the *point selection tool* from which distances between two selected points can be computed directly. The procedure for one such feature, in this case a wall, is illustrated in Figure 4.23. Here, two points on the top wall have been selected in the reconstruction and the corresponding wall in the BIM. The reconstruction is placed in the top part of the image and the sampled BIM is placed in the bottom part. See Appendix B.1 for visualization of the remaining measurements. Table 4.3 presents the measured distances and the deviation between the reconstruction and the BIM for multiple walls.

Table 4.3 The measured distances in the reconstruction and sampled building model.

Wall	$d_{reconstruction}$ [m]	d_{BIM} [m]	$\parallel \Delta d \parallel$ [m]
1	63.8795	63.8966	0.0171
2	16.3243	16.3181	0.0062
3	12.1694	12.0769	0.0925
4	11.7613	11.6002	0.1611
5	17.9853	18.0001	0.0148
6	17.8677	17.9677	0.1000
7	2.8410	2.8914	0.0504

Point to plane Ideally, points that are measured on a plane surface should lie on the plane that makes up the surface. In practice, the measurements are always subjected to noise to some extent and the platform from which the measurements are made is moving which in turn may introduce errors as a result of misalignment during the scan. In Figure 4.24 the selected geometry in the reconstruction is highlighted. The geometry is constituted by walls. The reason being that since the wall is a plane surface in the relative sense, the points that are measured on the wall should lie on a plane. For each selected section of the reconstruction, a plane is fitted to the points. The point-to-plane distance is then computed. Positive distance is in the direction of the normal of the plane towards the inward of the reconstruction. Figures 4.25, 4.26, 4.27 and 4.28 display the resulting histograms of the distances between the points and the corresponding plane fit. The mean distances and stan-

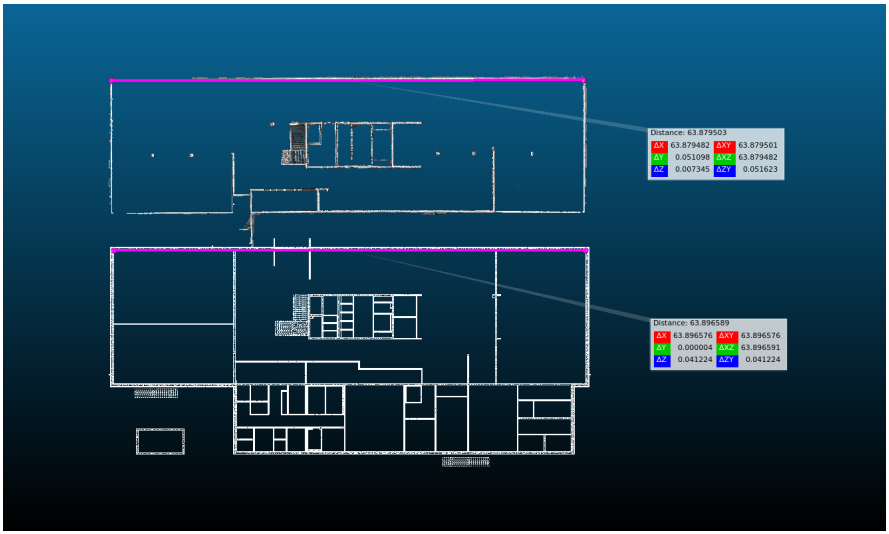

Figure 4.23 The measured distance in reconstruction (top) and sampled building model (bottom).

dard deviations between the points and the corresponding planes are presented in Table 4.4.

Table 4.4 The computed point-to-plane mean distances and standard deviation.

Wall	**Mean** [m]	**Std.dev** [m]
1	-0.0005	0.0263
2	-0.0007	0.0172
3	-0.0006	0.0246
4	-0.0006	0.0166

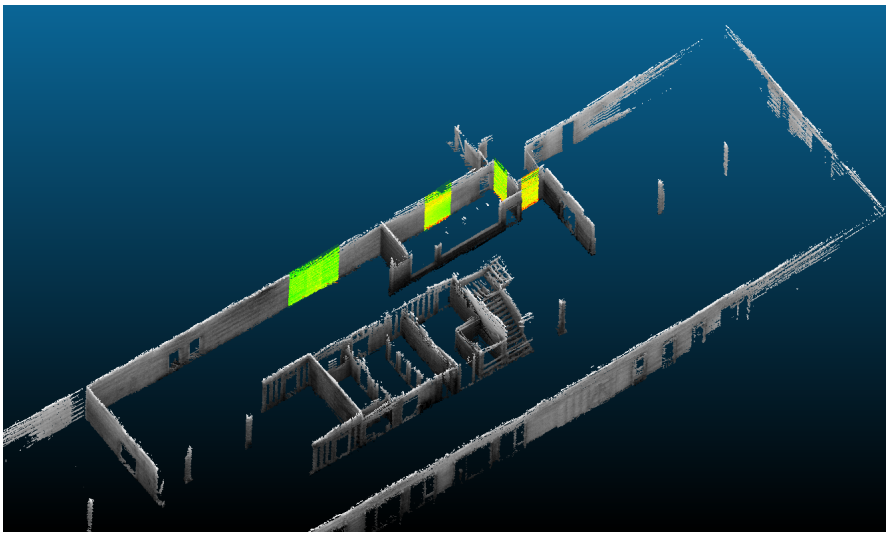

Figure 4.24 The selected surfaces (highlighted) in the reconstruction used for plane fitting.

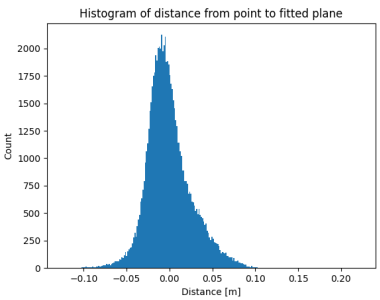

Figure 4.25 The resulting histogram of computing the point-to-plane distance for Wall 1.

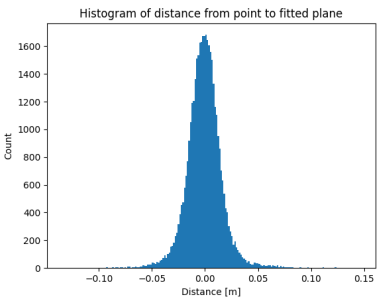

Figure 4.26 The resulting histogram of computing the point-to-plane distance for Wall 2.

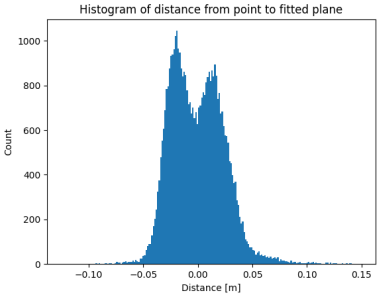

Figure 4.27 The resulting histogram of computing the point-to-plane distance for Wall 3.

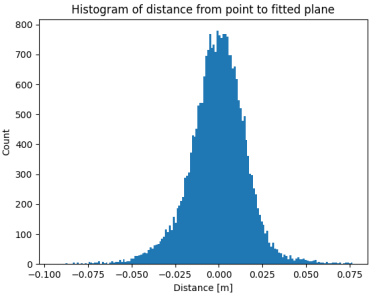

Figure 4.28 The resulting histogram of computing the point-to-plane distance for Wall 4.

5

Discussion

This chapter concludes the report with discussion regarding the prototype SLAM system and the results of the experiments conducted. To serve as a reminder, the three main objectives of the thesis were:

- Build a prototype visual SLAM system which incorporates the available sensor configuration on Spot.
- Benchmark the localization performance of the system by comparing against ground truth measurements.
- Evaluate the reconstuction accuracy.

The result of the work carried out during the thesis is a prototype system which integrates the passive features of Spot into the ROS framework. The system built and tested performs satisfactorily given the available sensor configuration. By leveraging the existing SLAM library RTABMap, the prototype system with the current sensor configuration and odometry creates dense reconstructions with relatively high accuracy. The loop closing capabilities of RTABMap adds robustness to the localization even in the presence of significant drift in the odometry. The drift is corrected by the added loop constraints in the pose graph optimization. The objectives set have thus been completed to some extent. The following sections will further present the reflections regarding the experiments and evaluation methods, results and suggested improvements.

5.1 Localization performance

First off, the unreliability of the ground truth for the localization experiments has to be addressed. There are multiple introduced uncertainties in the measurements and the following evaluation which may have a significant impact on the results of the localization performance. Since ground truth orientation could not be acquired due to the loss of measurements from one of the tracking systems, the yaw had to

be estimated from a single signal. The lacking orientation also meant that the poses had to be aligned by manually picking point correspondences and solving the absolute orientation problem. Further improvements to the experimental setup would be possible but could not be realized due to the lack of time. One such improvement would be to mount the antennas in a more appropriate configuration such that additional signal reflections would be minimized. Another improvement would be to either introduce a calibrated IMU (inertial measurement unit) to also measure pitch and roll or a third GPS setup.

When disregarding the flaws of the experimental setup for the localization experiments, the results seems to indicate that the pose graph approach is robust to significant odometry drift. The results from the localization experiments show that the loop closure detection of RTABMap corrects for large amounts of drift. When the localization is evaluated with the absolute trajectory error metric, the RMSE acheived were in the decimeter range in two of the experiments for paths longer than 100 meters.

5.2 Reconstruction accuracy

The LiDAR pointclouds registered to the estimated position of the robot from the pose graph produce seemingly accurate reconstructions. The alignment of the reconstruction to the sampled building model gives an indication of the scale accuracy. By visual inspection the scale of the reconstruction compared to the model gives the overall impression that the scale appears to be correct. This notion is further amplified by measuring certain geometry in the reconstruction and the corresponding geometry in the building model. The magnitude of the deviations range between centimeters and decimeters. The evaluation method may be considered flawed, mainly due to two aspects of the method used. The points used in the measurements are selected manually which increases the uncertainty. The reconstruction is also evaluated against the planned building model. Therefore, it is inconclusive whether or not the deviations are due to the manual selections of points, errors in the reconstruction or actual deviation from the planned model. A suggestion for a more reliable evaluation would have been to use a total station to acquire the measurements of distances between geometric features. However, this could not be realized within the scope of this thesis. Some consideration should also be taken to the fact that the measurements are made from a moving platform with the associated uncertainty in the positioning of the sensors.

The evaluation of the accuracy by computing the point-to-plane distance shows that for subsets of the reconstruction, the scan alignment of the points are at least as accurate as the LiDAR system that measured the points. Combined with the fact that the standard deviation of the point-to-plane distance falls within the specification of the range accuracy of the LiDAR, the assumption can be made that the scan alignment does not introduce any significant errors

6

Conclusion

The work carried out in this thesis has laid the necessary foundation to leverage existing SLAM solutions with the Spot robot platform to build dense reconstructions of the environment. The designed and tested prototype system integrates the sensor configuration of Spot into the ROS framework. Combined with the visual SLAM library RTABMap for robust localization with loop closing capabilities and dense camera and LiDAR reconstructions, the system provides means for online map building and data acquisition. The onboard depth cameras were deemed inadequate in providing any meaningful comparison to the building information model due to both the resolution and mounting on the platform. However, the visual information proved to be useful when utilized for localization purposes by detecting loop closures during mapping. The resulting LiDAR reconstructions provided comparable results to be evaluated against the planned building model and acheived centimeter accuracy with the current sensor configuration. The results from the evaluation of the reconstructions seem to indicate that the range accuracy of the LiDAR system is the limiting factor. Thus, the current solution is not sufficient to be used for site inspection. However, the reconstructions could still be useful for tracking the progress of ongoing constructions.

Further development

A feature which could improve the utility of the system in real world applications would be automatic registration of the reconstructions to the coordinate frame of the building model. This could be done by initializing the mapping in relation to a known landmark where the position of the landmark has been accurately measured in reference to the coordinate frame of the building model. The landmark could for instance be an aruco marker or any other easily identifiable type of landmark.

Bibliography

Arun, K., T. Huang, and S. Blostein (1987). "Least-squares fitting of two 3-D point sets". *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **PAMI-9**, pp. 698–700. DOI: 10.1109/TPAMI.1987.4767965.

Boverket (2018). *Kartläggning av fel, brister och skador inom byggsektorn.* URL: https://www.boverket.se/sv/om-boverket/publicerat-av-boverket/publikationer/2018/kartlaggning-av-fel-brister-och-skador-inom-byggsektorn/ (visited on 2022-01-24).

Campos, C., R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós (2021). "Orb-slam3: an accurate open-source library for visual, visual–inertial, and multimap slam". *IEEE Transactions on Robotics* **37**:6, pp. 1874–1890. DOI: 10.1109/TRO.2021.3075644.

Fraundorfer, F. and D. Scaramuzza (2012). "Visual odometry : part II: matching, robustness, optimization, and applications". *IEEE Robotics & Automation Magazine* **19**, pp. 78–90.

Grisetti, G., R. Kümmerle, C. Stachniss, and W. Burgard (2010). "A tutorial on graph-based SLAM". *IEEE Intelligent Transportation Systems Magazine* **2**:4, pp. 31–43. DOI: 10.1109/MITS.2010.939925.

Julier, S. and J. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. DOI: 10.1109/JPROC.2003.823141.

Labbé, M. and F. Michaud (2013). "Appearance-based loop closure detection for online large-scale and long-term operation". *IEEE Transactions on Robotics* **29**:3, pp. 734–745. DOI: 10.1109/TRO.2013.2242375.

Labbé, M. and F. Michaud (2019). "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". *Journal of Field Robotics* **36**:2, pp. 416–446. DOI: https://doi.org/10.1002/rob.21831. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21831. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831.

Leonard, J. and H. Durrant-Whyte (1991). "Simultaneous map building and localization for an autonomous mobile robot". In: *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*. Vol. 3, pp. 1442–1447. DOI: 10.1109/IROS.1991.174711.

Lourakis, M. (2016). "An efficient solution to absolute orientation". In: *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3816–3819. DOI: 10.1109/ICPR.2016.7900229.

Lowe, D. G. (2004). "Distinctive image features from scale-invariant keypoints". *International Journal of Computer Vision* **60**, pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94.

Lynch, K. M. and F. C. Park (2017). *Modern Robotics: Mechanics, Planning, and Control*. 1st. Cambridge University Press, USA. ISBN: 1107156300.

Olsson, C. (2021). *Computer Vision - Lecture notes*. LTH, Lund University.

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng (2009). "ROS: an open-source robot operating system". In: *ICRA 2009*. Vol. 3.

Sivic, J. and A. Zisserman (2003). "Video google: a text retrieval approach to object matching in videos". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. Vol. 2, pp. 1470–1477. DOI: 10.1109/ICCV.2003.1238663.

Taketomi, T., H. Uchiyama, and S. Ikeda (2017). "Visual SLAM algorithms: a survey from 2010 to 2016". *IPSJ Transactions on Computer Vision and Applications* **9**. DOI: 10.1186/s41074-017-0027-2.

Thrun, S. and J. J. Leonard (2008). "Simultaneous localization and mapping". In: Siciliano, B. et al. (Eds.). *Springer Handbook of Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 871–889. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_38. URL: https://doi.org/10.1007/978-3-540-30301-5_38.

Zhang, Z. and D. Scaramuzza (2018). "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7244–7251. DOI: 10.1109/IROS.2018.8593941.

A

Localization

This appendix contains the python code written for solving the absolute orientation problem as described in Section 2.4. The appendix also contains the resulting reconstructions from the localzation experiments presented in Section 4.1.

A.1 Python implementation for solving the absolute orientation problem

```
import numpy as np

gps = np.loadtxt('gps_corr.csv', delimiter = ',')
graph = np.loadtxt('graph_corr.csv', delimiter=',')
xm = np.array([np.mean(graph[:,0]),
    np.mean(graph[:,1]),np.mean(graph[:,2])])
ym = np.array([np.mean(gps[:,0]),
    np.mean(gps[:,1]),np.mean(gps[:,2])])
qn = np.transpose(graph - xm)
qn_prim = np.transpose(gps - ym)

H = np.zeros([3,3])
for i in range(np.size(qn,1)):

    qn_s = np.reshape(qn[:,i],(3,1))
    qn_prim_s = np.reshape(qn_prim[:,i],(1,3))
    res = np.matmul(qn_s,qn_prim_s)
    H = H + res

H = 1/np.size(qn,1) * H
u,s,v = np.linalg.svd(H)
R = np.matmul(np.transpose(v),np.transpose(u))
t = ym - np.matmul(R,xm)
```

A.2 Reconstructions

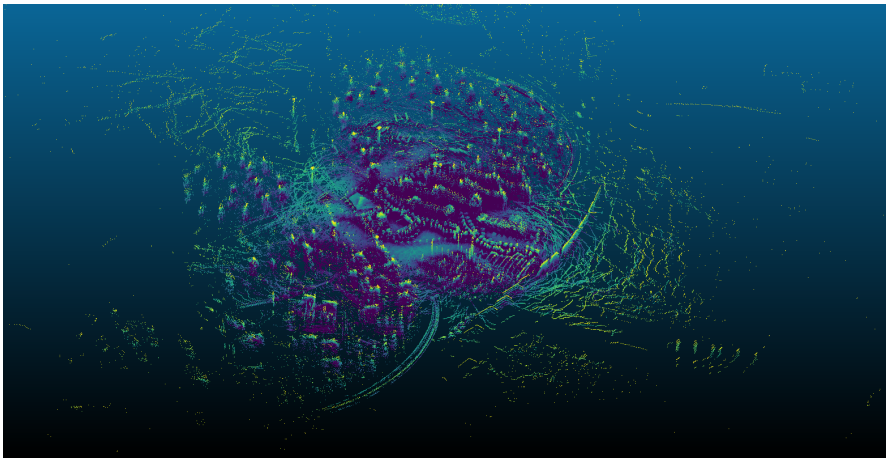

Figure A.1 The resulting LiDAR reconstruction from the first experiment at Brunnshögs stationstorg visualized in CloudCompare.

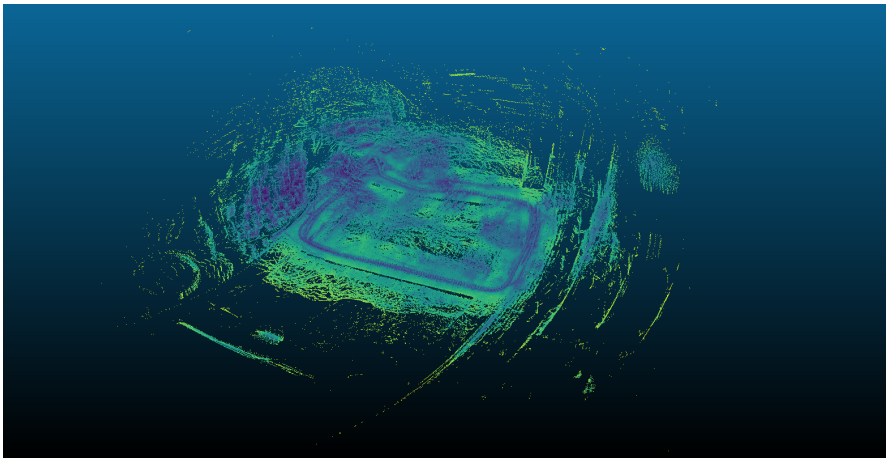

Figure A.2 The resulting LiDAR reconstruction from the second experiment at Brunnshögs stationstorg visualized in CloudCompare.

B

Vipan

This appendix contains the figues illustrating the selection of corresponding points in the sampled building information model and reconstruction as described in Section B.1.

B.1 Measured distances

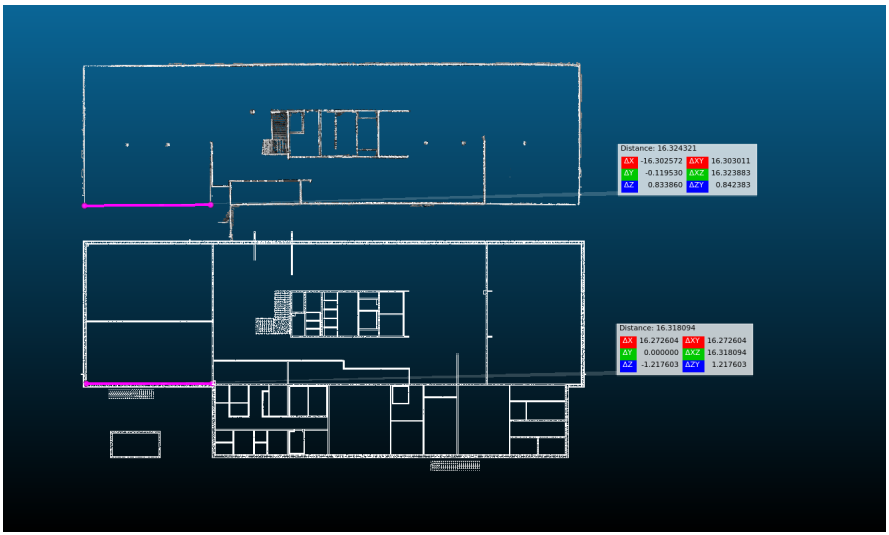

Figure B.1 Measured distance in reconstruction and BIM.

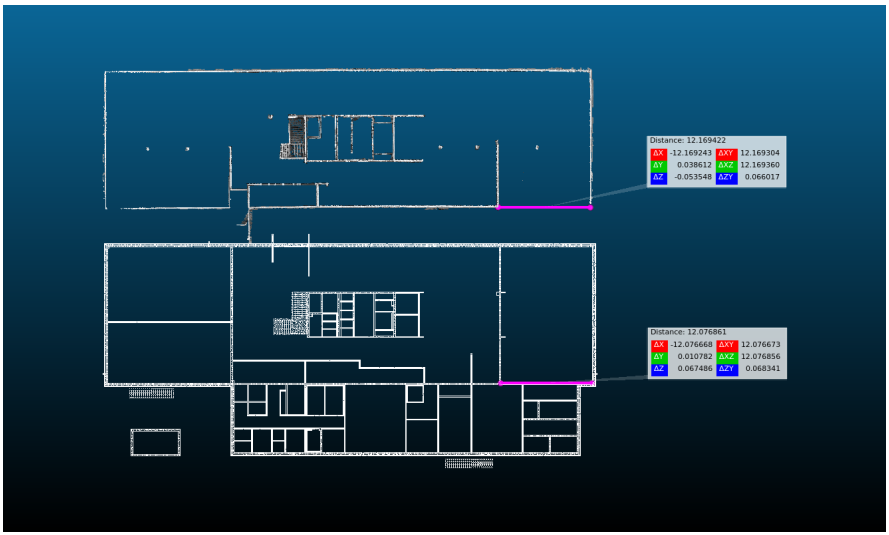

Figure B.2 Measured distance in reconstruction and BIM.

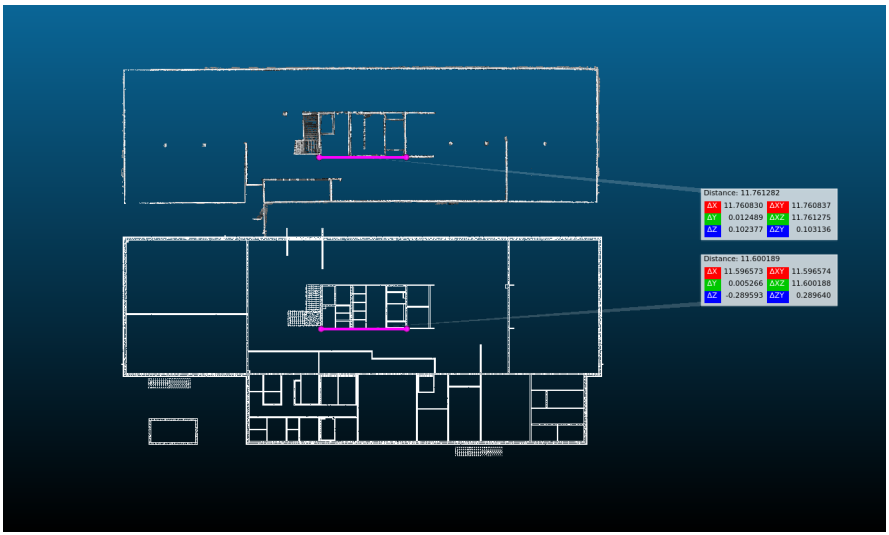

Figure B.3 Measured distance in reconstruction and BIM.

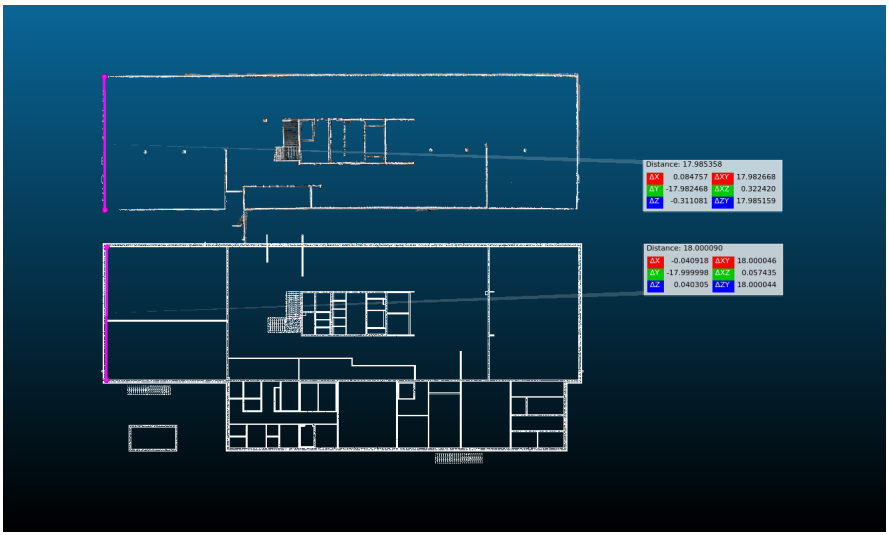

Figure B.4 Measured distance in reconstruction and BIM.

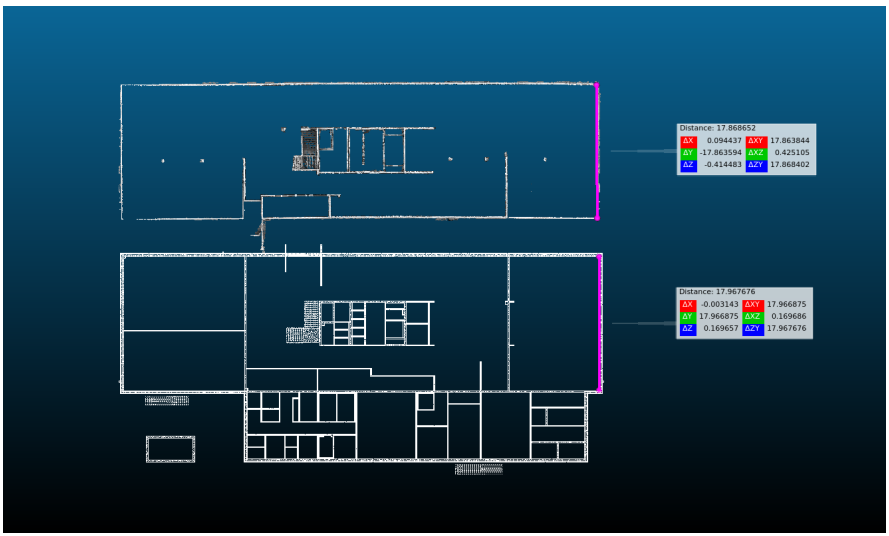

Figure B.5 Measured distance in reconstruction and BIM.

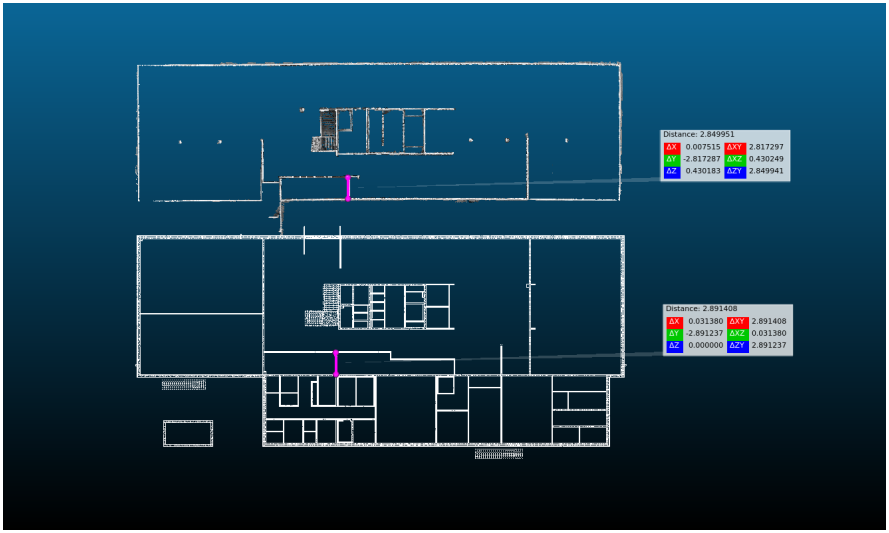

Figure B.6 Measured distance in reconstruction and BIM.

Lund University
Department of Automatic Control
Box 118
kontraktSE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
May 2022

Document Number
TFRT-6158

Author(s)
Ola Nilsson

Supervisor
Maike Klöckner, Dept. of Computer Science, Lund University, Sweden
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Anton Cervin, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Building dense reconstructions with SLAM and Spot

Abstract

Having access to dense reconstruction of ongoing building constructions provides insight into the building process and could serve as both a tool for error detection and documentation of the actual outcome. In this thesis, Spot, the quadruped robot designed by Boston Dynamics is evaluated as a platform for site inspection. The result of the thesis was a prototype system built on top of the ROS framework which integrates the sensors of the robot platform. Combined with the visual SLAM library RTABMap for robust localization with loop closing capabilities and dense camera and LiDAR reconstructions, the system provides means for online map building and data acquisition. The localization performance achieved positional RMSE in the decimeter range for paths spanning further than 100 meters. The resulting LiDAR reconstructions provided comparable results to be evaluated against the planned building model and acheived centimeter accuracy with the current sensor configuration.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-61

Recipient's notes

Security classification

http://www.control.lth.se/publications/