TÉCNICO
LISBOA

Topological Mapping for Sentinel Mobile Robots

Miguel da Cunha Ferreira dos Anjos

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor

Professor José António da Cruz Pinto Gaspar

Examination Committee

Chairperson: Professor António Manuel Raminhos Cordeiro Grilo
Supervisor: Professor José António da Cruz Pinto Gaspar
Member of the Committee: Professor Alberto Manuel Martinho Vale

November 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Agradecimentos

A escrita e entrega deste documento marca a passagem de uma fase da minha vida extremamente importante para o meu desenvolvimento profissional, intelectual e pessoal. É com algum orgulho e bastante satisfação que termino esta etapa da minha vida.

O meu percurso académico não foi linear. É de certa forma irónico, por isso, que a minha dissertação seja na área da robótica móvel que lida com não linearidades em praticamente todas as partes constituintes. Houveram altos e baixos neste percurso, mas sinto que apenas valorizam o alcance desta meta final, e que acrescentaram, e muito, ao meu crescimento como pessoa.

Quero deixar uma palavra forte de agradecimento ao Professor José Gaspar por todo o acompanhamento e desafios que me foi colocando, assim como por toda a ajuda que sempre me forneceu ao longo deste último passo que é a dissertação.

Quero deixar um muito forte agradecimento à minha namorada, Vanda, que sempre acreditou em mim e na minha capacidade de caminhar para o objetivo desde que me conheceu, apesar das fracas circunstâncias em que me encontrava.

Quero também deixar um agradecimento muito forte à minha mãe, Isabel, à minha madrasta, Célia, à minha irmã Carolina e ao meu Avô Carlos.

Agradeço, por último e de forma especial, ao meu pai, António, que sempre me proporcionou tudo o que vê como ajuda ao sucesso e me continuou a apoiar apesar de, por vezes, lhe ter dificultado a fé no meu caminho.

Foram todos uma forte ajuda e motivação para que conseguisse alcançar este objetivo.

A todos, um forte obrigado.

Resumo

A disponibilidade e utilização de robôs móveis autónomos na vida quotidiana está a tornar-se mais real a cada dia que passa. É cada vez mais importante conceber sistemas para que os robôs móveis interajam a um nível mais abstracto com os humanos quando se trata de navegação.

Nesta dissertação, propomos uma metodologia de mapeamento topológico baseada no agrupamento de estruturas 3D para a criação de nós, com o objectivo de realizar uma contribuição para a navegação e compreensão semântica.

Detalhamos tanto os componentes de software como de hardware da construção de um robô móvel sentinela. O robô está equipado com um Raspberry Pi como sua principal unidade computacional, uma câmara monocular e uma unidade de medição inercial.

Utilizamos este robô móvel para avaliar o processo de criação de mapas topológicos proposto e o seu sucesso no reconhecimento de um local anteriormente mapeado. Para além de utilizarmos os nossos próprios conjuntos de dados adquiridos, utilizamos também um conjunto de dados públicos para uma das nossas experiências.

Palavras chave: Mapeamento Topológico, Agrupamento 3D, Robô Móvel, Raspberry Pi, Navegação Semântica.

Abstract

The availability and utilization of autonomous mobile robots in everyday life is becoming more of reality with each passing day. It is increasingly more important to design systems for mobile robots to interact on an abstract level with humans when it comes to navigation.

In this dissertation, we propose a methodology of topological mapping based on clustering 3D structures for node creation, with the objective of contributing towards semantic navigation and understanding.

We detail both the software and hardware components of how we built a sentinel mobile robot. The robot is equipped with a Raspberry Pi as its main computational unit, a monocular camera and an inertial measurement unit.

We utilize this mobile robot to evaluate our proposed topological map creation process and how successful it is at recognizing a formerly mapped location. In addition to using our own acquired datasets we also use a public dataset for one our experiments.

Keywords: Topological Mapping, 3D clustering, Mobile Robot, Raspberry Pi, Semantic Navigation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile autonomous robots are one of the most researched topics in recent years with commercial designs already available for both industrial settings, e.g. Waypoints's Vector, and for home settings, e.g. the vacuum Roomba from iRobot. The most common sensor deployed for accurate navigation is the LiDAR sensor [1] since it provides accurate depth information which facilitates estimating the robot's pose. The excellent quality of LiDAR sensors implies, however, very high hardware costs.

Nowadays, the availability of low cost cameras and inertial measurement units (IMU) means developing autonomous robots based on this sensor suite is very promising. If an accurate and real-time performing platform can be developed at a low cost, it might open the door for a future in which most people have mobile robots that do surveillance, carry objects from room to room autonomously and more. In a world where most people deal with robots, it seems imperative that the robots possess a human-like understanding of the world as robust navigation.

1.1 Related Work

A low-cost mobile system capable of navigation using a camera and an IMU has been explored previously in the literature. Fusing these sensors started with only estimating the local positioning of the platform without loop closure, i.e. visual-inertial odometry (VIO), as in the works [16][24] .

For a more complete system with global consistency, and potentially multi-session mapping, SLAM is the next step. Filter-based solutions have been proposed in [25] [34] that include loop

[1] e.g. Velodyne's LiDAR sensors are considered market leaders having been part of the Waymo/Google project autonomous car.

closure and as such are capable of global consistency. Optimization-based algorithms seem to be the current trend of the art, with [38], [30], [31], [15], [39] and [5] all developed since 2017 and with increasing performance. In particular, [5] can be said to be representative of the state of the art for visual and visual-inertial SLAM systems, capable of multi-session navigation, with a multiple map system with short, medium and long term recall capabilities.

These state-of-the-art algorithms have very high metric accuracy, however no semantic meaning of places is attempted [21]. For a robot to navigate homes using the same paradigm of humans, assignment of semantic meaning to places has to occur. The first step towards meaning assignment is the creation of topological maps, which has been explored for many years. Topomap [3] uses visual information to create topological maps and navigate them. From these topological maps, semantic meaning can be assigned to nodes of the map with research on this topic conducted recently in [1] and [6].

1.2 Problem Formulation and Thesis Approach

In this thesis we aim to address the first step in our view to enabling mobile robots to communicate with humans on the same abstract level when it comes to navigation, which is topological mapping.

As an example, humans navigate home environments not with reference to a frame, but by assigning meaning to locations, such as 'kitchen', and recognizing they are there or if they are in another location. For an autonomous system to be able to assign semantic meaning to locations it must first be able to differentiate them in some way.

Topological maps are a valuable tool for this since they provide discrete states that can be assigned meaning.

We propose a topological mapping framework that takes the output of a SLAM system that provides 3D structure and creates a scene graph. This scene graph is composed of nodes that map to 3D clusters of the environment. To evaluate if node recognition in this framework is successful we present several experiments where the robot navigates for a second time in a particular scene and matches the nodes being created for the current navigation to the first one.

A robot platform similar to the one developed in [27] was developed and used in these experiments. It uses a monocular camera and an inexpensive IMU.

1.3 Objectives and Contributions

In this work, we aim to design and build a sentinel mobile robot equipped with an IMU and a monocular camera. We also aim to design a topological mapping and matching system and test it in the developed platform.

To accomplish this we build our software system on top of several public software components. We utilize either a SLAM filter [25] or a structure from motion software, VisualSFM [2], to estimate the geometric structure of the environment. We utilize the Generalized-ICP algorithm [35] for registering the pointclouds of two scenes. We utilise the DBSCAN [14] algorithm for finding 3D clusters.

We combine all of these components such that a scene graph is created for a given navigation in a custom software system of our own. We also build a custom matching system for matching the scene graphs of two navigated scenes.

1.4 Thesis Structure

Chapter 1 introduces the problem we approach in this thesis and how it relates to the larger literature body in the mobile robotics field.

Chapter 2 presents the state of the art of several components that are necessary for a mobile system capable of solving the approached problem.

Chapter 3 describes the mobile sentinel robot developed, both in terms of hardware and software infrastructure.

Chapter 4 is a description of the proposed methodology for creating topological maps and recognizing locations, we also present the evaluation process which we use for ascertaining if location recognition was successful.

Chapter 5 provides an overview of the main experiments performed as well of the results obtained. A critical analysis of interesting observations is provided. Different experiment setups are shown to evaluate the system at different degrees of difficulty and different conditions.

Chapter 6 summarizes the work developed and its main achievements. Possible avenues for future work and system improvement are also provided.

[2]VisualSFM is a GUI application for 3D reconstruction using structure from motion (SFM). `http://ccwu.me/vsfm/index.html`

Chapter 2

Background and State of the Art

Since our approach relies on SLAM systems we provide background on the main approaches that utilize our sensor suite. There are two main approaches to solving the visual and, by extension, the visual-inertial (VI) SLAM problem. There is the classical way of solving V-SLAM which is a filter-based solution, solved for real-time applications in [11] and other, that estimates both the system's pose and the tracked landmarks positions by enlarging the state vector and then using a filter such as the Extended Kalman Filter (EKF) to predict the evolution of the entire state. There are particle filters that model the state as a stochastic process and sample the state space in order to estimate current pose and map state [22]. And there is the key-frame bundle adjustment (BA) strategy used by [5][15] and others, based on optimization algorithms, which find the pose of the system by minimizing the pose error between several keyframes taking into account the movement model. This later strategy is capable of better accuracy than the former but demands higher computation time [21].

2.1 Filter-Based Visual and Visual-Inertial SLAM

The first visual SLAM algorithm capable of real time performance was proposed in [11] and updated in [8]. The following is an explanation of the mathematics behind MonoSLAM, which will also serve the purpose of explaining the motion model used in most state-of-the-art algorithms that deal in the euclidean plane, as opposed to manifolds in Lie algebra which is another common representation in today's state-of-the-art algorithms [16] [25]. It should also serve the purpose of representing a template for how filter-based SLAM algorithms work. It uses an extended Kalman filter (EKF) for state estimation.

MonoSLAM, Assumption of Smooth Motion

The model used is a statistical motion model, that assumes constant velocity and constant angular velocity, with undetermined accelerations having a Gaussian profile. This assumption imposes smoothness on the expected camera motion, i.e., very large accelerations are relatively unlikely. At each time step unknown linear acceleration $\mathbf{a}^{\mathbf{W}}$ and unknown angular acceleration $\alpha^{\mathbf{W}}$ processes of zero mean and Gaussian distribution cause an impulse of velocity and angular velocity:

$$\mathbf{n} = \begin{bmatrix} \mathbf{V}^W \\ \Omega^R \end{bmatrix} = \begin{bmatrix} \mathbf{a}^{\mathbf{W}} \Delta t \\ \alpha^{\mathbf{W}} \Delta t \end{bmatrix} \tag{2.1}$$

The covariance matrix of noise vector $\mathbf{n}$ is assumed to be diagonal, representing uncorrelated noise in all linear and rotational components.

State update

The state update produced is:

$$\mathbf{f_v} = \begin{bmatrix} \mathbf{r}^W_{new} \\ \mathbf{q}^{WR}_{new} \\ \mathbf{v}^W_{new} \\ \omega^R_{new} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^W + (\mathbf{v}^W + \mathbf{V}^W)\Delta t \\ \mathbf{q}^{WR} \times \mathbf{q}((\omega^R + \mathbf{\Omega}^R)\Delta t) \\ \mathbf{v}^W + \mathbf{V}^W \\ \omega^R + \mathbf{\Omega}^R \end{bmatrix} \tag{2.2}$$

Here, the notation $\mathbf{q}^{WR} \times \mathbf{q}((\omega^R + \mathbf{\Omega}^R)\Delta t)$ denotes the quaternion trivially defined by the angle-axis rotation vector $(\omega^R + \mathbf{\Omega}^R)\Delta t$. With $\mathbf{r}^W$ representing the camera position in the world frame, with $\mathbf{q}^{WR}$ being the quaternion defining its orientation relative to the world frame, with $\mathbf{v}^W$ defining its linear velocity and ω^R defining its angular velocity.

In the EKF, the new state estimate $\mathbf{f}_v(\mathbf{x}_v, \mathbf{u})$ must be accompanied by the increase in state uncertainty (process noise covariance) Q_v for the camera after this motion. We find Q_v via the Jacobian calculation:

$$Q_v = \frac{\delta \mathbf{f}_v}{\delta \mathbf{n}_v} \mathbf{P}_n \frac{\delta \mathbf{f}_v}{\delta \mathbf{n}_v}^\top \tag{2.3}$$

where P_n is the covariance of noise vector $\mathbf{n}$. EKF implementation also requires calculation of the Jacobian $\frac{\delta \mathbf{f}_v}{\delta \mathbf{x}_v}$. The rate of growth of uncertainty in this motion model is determined by the size of P_n and setting these parameters to small or large values defines the smoothness of the motion expected. With small P_n, a smooth motion with small accelerations is expected, and would be well placed to track motions of this type, but unable to cope with sudden rapid movements. High P_n means that the uncertainty in the system increases significantly at each time step and while this gives the ability to cope with rapid accelerations the very large uncertainty means that a lot of good measurements must be made at each time step to constrain estimates.

Measurement Model

Active Feature Measurement and Update

This section concerns to the measurement of features that are already in the SLAM map. Each observed feature imposes a constraint between the camera location and the corresponding map feature. Observation of a point $\mathbf{y}_i(\mathbf{x}_i)$ defines a ray coded by a directional vector in the camera frame $\mathbf{h}^C = \begin{bmatrix} h_x & h_y & h_z \end{bmatrix}$. For the standard parametrization, $\mathbf{x}_i = \begin{bmatrix} X_i & Y_i & Z_i \end{bmatrix}^\top$:

$$\mathbf{h}_L^R = \mathbf{R}^{RW}(\mathbf{y}_i{}^W - \mathbf{r}^W) \tag{2.4}$$

With a perspective camera, the position (u, v) at which the feature would be expected to be found in the image is found using the standard pinhole model:

$$\mathbf{h}_i = \begin{bmatrix} u \\ \\ v \end{bmatrix} = \begin{bmatrix} u_0 - fk_u \frac{\mathbf{h}_{Lx}^R}{\mathbf{h}_{Lz}^R} \\ \\ v_0 - fk_v \frac{\mathbf{h}_{Ly}^R}{\mathbf{h}_{Lz}^R} \end{bmatrix} \tag{2.5}$$

where fk_u, fk_v, u_0, and v_0 are the standard camera calibration parameters.

The Jacobians of this two-step projection function with respect to camera and feature positions are also computed. These allow calculation of the uncertainty in the prediction of the feature image location, represented by the symmetric 2×2 innovation covariance matrix S_i, with radially unwarped perspective-projected coordinates, $\mathbf{u}_d = (u_d, v_d)$, that are the final predicted image position:

$$S_i = \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{x}_v} P_{xx} \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{x}_v}^\top + \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{x}_v} P_{xy_i} \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{y}_i}^\top + \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{y}_i} P_{y_i x} \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{x}_v}^\top + \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{y}_i} P_{y_i y_i} \frac{\partial \mathbf{u}_{di}}{\partial \mathbf{y}_i}^\top + R \tag{2.6}$$

The constant noise covariance R of measurements is taken to be diagonal with magnitude determined by image resolution. S_i represents the shape of a 2D Gaussian probability density function (PDF) over image coordinates and choosing a number of standard deviations (gating, normally at 3) defines an elliptical search window within which the feature should lie with high probability. Correlation searches always occur within gated search regions, maximizing efficiency and minimizing the chance of mismatches. S_i is also a measurement of the information content expected of a measurement. This means feature searching with high S_i will provide more information [10] about estimates of camera and feature positions. In the case that many candidate measurements are available those with high innovation covariance are selected, limiting the maximum number of feature searches per frame to the 10 or 12 most informative in the original paper.

Conclusion

MonoSLAM was an advancement in terms of visual real-time SLAM. It proved that with enough ingenuity it was possible to use a single camera to do SLAM. Since the original paper was published, a plethora of research has been done on this topic, both in the direction of improving the filter-based SLAM like in [11] [24][25][19] , as well as in the direction of optimization algorithms, [5][15], which tend to be more precise at the cost of computation time [21].

The motion model, state representation and filter strategy discussed in this section are representative of the structure used in most filter-based strategies of the state of the art approaches, with the exception of the state which has another common representation in the form of a Lie group representation [25]. There are new and improved filters which generally add or remove constraints to either estimate better or faster the state, but the probabilistic approach remains the same.

2.2 Fusion of Inertial and Visual inputs

The addition of a different sensor's information is very useful for better estimating the already observable variables of the state, but also to introduce new observable state variables in case such is possible.

Inertial measurement units provide information on the acceleration, angular rate and, if the IMU has a magnetometer, the surrounding magnetic field. A monocular camera provides much information about the world, but one state variable that is not observable is the true scale.

By fusing these two sensors it is possible to estimate this variable while also improving the estimates of all other state variables while solving SLAM.

Fusing both of these sensors data is done in several ways in the literature [21]. It is possible to classify all of them into one of two groups: loosely-coupled or tightly-coupled.

The loosely-coupled fusion, in either filtering or optimization-based estimation, processes the visual and inertial measurements separately to infer their own motion constraints and then fuse these constraints. In contrast the tightly-coupled approaches directly fuse the visual and inertial measurements within a single process.

Tightly-coupled fusion seems to yield better results and researchers postulate the decoupling of visual and inertial constraints of loosedly-coupled fusion results in information loss [21].

There is a need of preintegrating inertial measurements before fusing them with visual information and it comes from the different sample rates of the IMU and camera, +100Hz and ~30Hz respectively, which would require that the filter computed updates at high rates or that the system stored a large number of inertial observations to process them in batch when new visual information is available.

By first integrating these observations they can be treated as a single observation in the filter, reducing the problems listed previously.

First introduced in [24], where the authors employed the discrete integration of the inertial measurement dynamics in a local frame of reference, preventing the need to reintegrate the state dynamics at each optimization step. However, due to the use of Euler angles in the orientation representation, it suffers from singularities.

In [16] a different orientation representation based on manifolds is introduced which presents a singularity-free orientation representation on the $SO(3)$ manifold.

Recently a closed form solution for preintegration on manifold was presented in [12].

After preintegration one has access to, for each camera frame, a single IMU measure which can either be used for filter-based SLAM solvers or optimization based solvers.

2.3 Unscented Kalman Filter

The Brossard et al. [25] filter is a foundation of our project as the chosen SLAM system utilized. It is an Unscented Kalman filter on Lie groups that fuses camera and IMU measurements together in order to estimate the system's state. We discuss the system models that this filter utilizes and will also provide an introduction to the Lie theory required for an overview of these models and the filter in Appendix B and provide here a summary of the filter.

Filter Summary

The filter architecture is presented as:

$$\text{state}\begin{cases}\chi = \mathbf{exp}(\xi)\overline{\chi} \\ b_n = \overline{b}_n + \tilde{b}\end{cases}, \begin{bmatrix}\xi \\ \tilde{b}\end{bmatrix} \sim \mathcal{N}(0,\, \mathbf{P}_n), \tag{2.7}$$

$$\text{dynamics}\begin{cases}\chi_n, b_n = f(\chi_{n-1}, u_n - b_{n-1}, n_n)\end{cases}, \tag{2.8}$$

$$\begin{cases}\mathbf{Y}_n \ = \ [\mathbf{y}_1{}^\top \ \dots \ \mathbf{y}_p{}^\top] \ := \ \mathbf{Y}(\chi_n, \omega_n) \\ \mathbf{y}_i \quad \text{according to eq.B.11}, \quad i = 1, \dots, p\end{cases}, \tag{2.9}$$

with $(\overline{\chi}_n, \overline{b}_n)$ as the mean estimate of the state at time n, $\mathbf{P}_n \in \mathbb{R}^{(15+3p)\times(15+3p)}$ as the covariance matrix that represents the state's uncertainties $(\xi, \tilde{b})$, and vector $\mathbf{Y}_n$ that contains the observations of the p landmarks with associated Gaussian noise $\omega_n \sim \mathcal{N}(0,\, \mathbf{W})$.

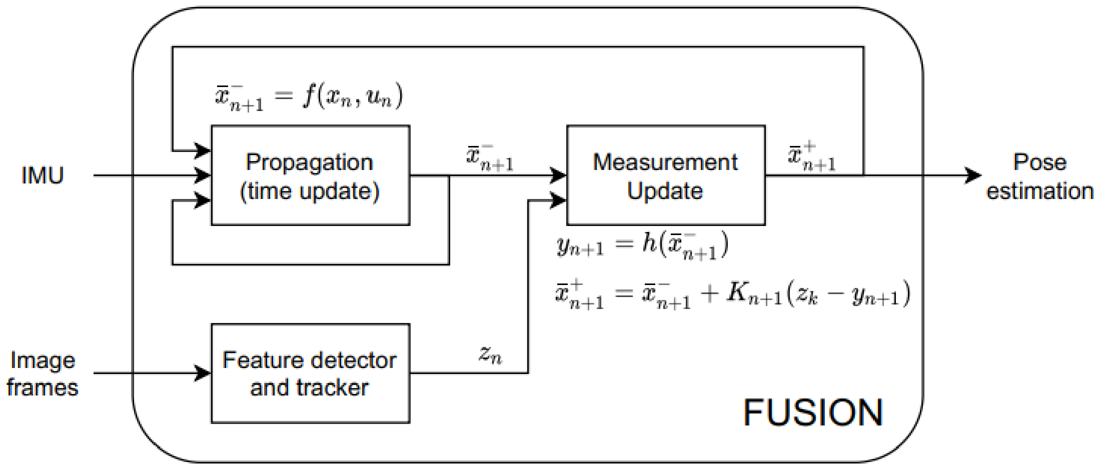

$\bar{x}^-_{n+1} = f(x_n, u_n)$
IMU
Propagation (time update)
$\bar{x}^-_{n+1}$
Measurement Update
$\bar{x}^+_{n+1}$
Pose estimation
$y_{n+1} = h(\bar{x}^-_{n+1})$
$\bar{x}^+_{n+1} = \bar{x}^-_{n+1} + K_{n+1}(z_k - y_{n+1})$
Image frames
Feature detector and tracker
z_n
FUSION

Figure 2.1: Block diagram of Unscented Kalman filter on Lie groups used in our work. From [27]

2.4 Optimization on Visual and Visual-Inertial SLAM

Optimization for local pose estimation is a popular method in recent research for solving SLAM [30] [15] [39] [5].

It provides better overall accuracy compared to filter-based pose estimation because relinearization is performed during each iteration step. There is no inconsistency issue in optimization based SLAM and thus the quality of the estimate is higher than that of EKF-SLAM, which linearizes only once leading to higher linearization errors.

The optimization for local pose estimation problem can be formulated as a non-linear least squares (NLLS) problem with parameters to be estimated including all the robot poses and all the feature positions. This NLLS problem can be solved by iterative methods such as Gauss-Newton or Levenberg - Marquardt.

ORBSLAM3

One state of the art complete SLAM system that uses an optimization approach is ORBSLAM3 [5], and the following is the author's formulation for this optimization problem.

The estimated state includes the body pose $T_i = [R_i, p_i] \in SE(3)$, the velocity v_i in the world frame and the gyroscope and accelerometer biases, b_{gi} and b_{ai}, which are assumed to evolve according to a Brownian motion. This leads to the state vector:

$$\mathcal{S}_i \doteq \{\mathbf{T}_i, \mathbf{v}_i, \mathbf{b}_i^g, \mathbf{b}_i^a\}. \tag{2.10}$$

The IMU measurements between two consecutive frames i and $i+1$ are preintegrated following the theory developed in [24] and formulated on manifolds in [16]. The preintegrated rotation, velocity, and position measurements, denoted as $\Delta R_{i,i+1}$, $\Delta v_{i,i+1}$ and $\Delta p_{i,i+1}$ as well as a covariance matrix $\Sigma I_{i,i+1}$ are obtained for the whole measurement vector. Given these preintegrated terms and states S_i and S_{i+1}, the definition of inertial residual $\mathbf{r}_{\mathcal{I}_{i,i+1}}$ is adopted from [16]

$$\begin{aligned}
\mathbf{r}_{\mathcal{I}_{i,i+1}} &= [\mathbf{r}_{\Delta \mathrm{R}_{i,i+1}}, \mathbf{r}_{\Delta \mathrm{v}_{i,i+1}}, \mathbf{r}_{\Delta \mathrm{p}_{i,i+1}}] \\
\mathbf{r}_{\Delta \mathrm{R}_{i,i+1}} &= \mathrm{Log}\left(\Delta \mathbf{R}_{i,i+1}^{\mathrm{T}} \mathbf{R}_i^{\mathrm{T}} \mathbf{R}_{i+1}\right) \\
\mathbf{r}_{\Delta \mathrm{v}_{i,i+1}} &= \mathbf{R}_i^{\mathrm{T}} \left(\mathbf{v}_{i+1} - \mathbf{v}_i - \mathbf{g}\Delta t_{i,i+1}\right) - \Delta \mathbf{v}_{i,i+1} \\
\mathbf{r}_{\Delta \mathrm{p}_{i,i+1}} &= \mathbf{R}_i^{\mathrm{T}} \left(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i \Delta t_{i,i+1} - \frac{1}{2}\mathbf{g}\Delta t^2\right) - \Delta \mathbf{p}_{i,i+1}
\end{aligned} \tag{2.11}$$

where $Log : SO(3) \rightarrow \mathbb{R}^3$ maps from the Lie group to the vector space. Together with inertial residuals, the authors also use reprojection errors $\mathbf{r}_{ij}$ between frame i and 3-D point j at position $\mathbf{x}_j$

$$\mathbf{r}_{ij} = \mathbf{u}_{ij} - \Pi\left(\mathbf{T}_{\mathrm{CB}} \mathbf{T}_i^{-1} \oplus \mathbf{x}_j\right) \tag{2.12}$$

where $\Pi : \mathbb{R}^3 \to \mathbb{R}^n$ is the projection function for the corresponding camera model, $\mathbf{u}_{ij}$ is the observation of point j at image i, having a covariance matrix Σ_{ij}, $\mathbf{T}_{\text{CB}} \in SE(3)$ stands for the rigid transformation from body-IMU to camera (left or right), known from calibration, and $\oplus$ is the transformation operation of $SE(3)$ group over $\mathbb{R}^3$ elements.

Combining inertial and visual residual terms, visual–inertial SLAM can be posed as a keyframe-based minimization problem [23]. Given a set of $k+1$ keyframes and its state $\bar{\mathcal{S}}_k \doteq \{\mathcal{S}_0 \dots \mathcal{S}_k\}$ and a set of l 3-D points and its state $\mathcal{X} \doteq \{\mathbf{x}_0 \dots \mathbf{x}_{l-1}\}$, the visual–inertial optimization problem can be stated as follows:

$$\min_{\bar{\mathcal{S}}_k, \mathcal{X}} \left(\sum_{i=1}^{k} \left\| \mathbf{r}_{\mathcal{I}_{i-1,i}} \right\|^2_{\Sigma^{-1}_{\mathcal{I}_{i,i+1}}} + \sum_{j=0}^{l-1} \sum_{i \in \mathcal{K}^j} \rho_{\text{Hub}} \left(\left\| \mathbf{r}_{ij} \right\|_{\Sigma^{-1}_{ij}} \right) \right) \tag{2.13}$$

where K^j is the set of keyframes observing 3-D point j. For the reprojection error, the authors use a Huber kernel ρ_{Hub} to reduce the influence of spurious matchings, while for inertial residuals, it is not needed since miss-associations do not exist.

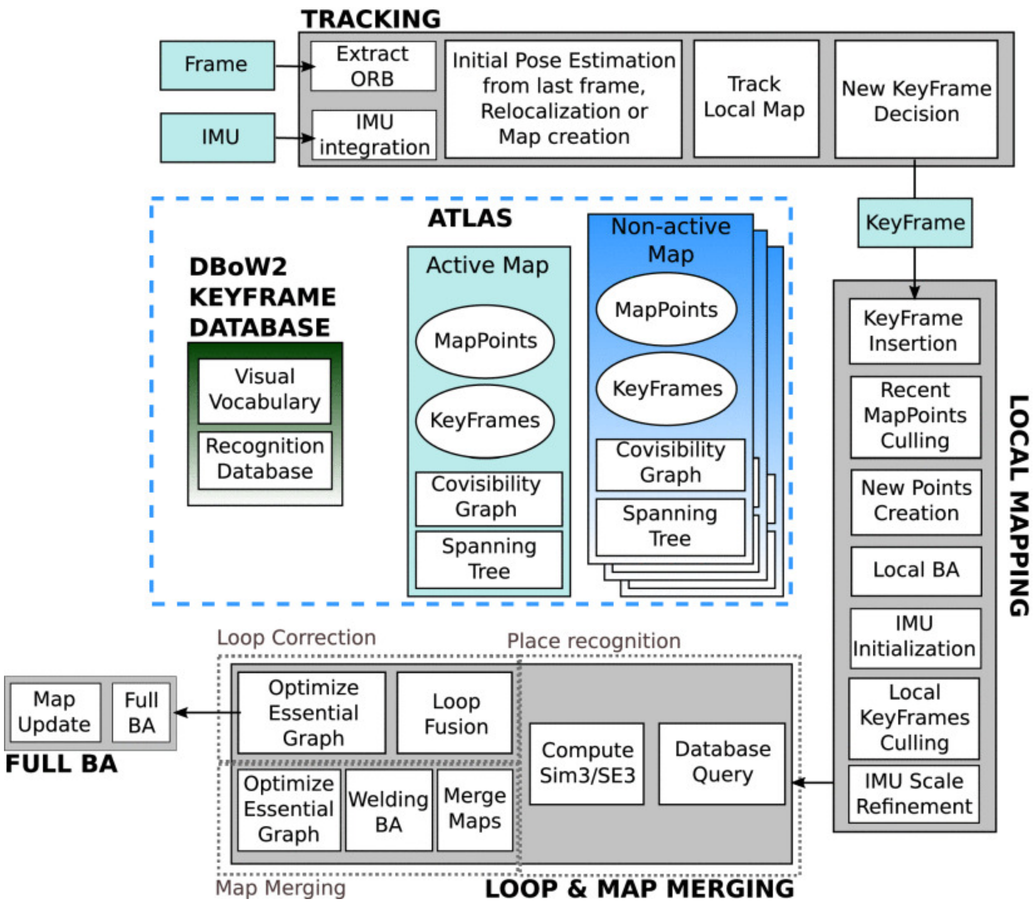


Figure 2.2: Full ORBSLAM3 system overview, from [5]

Loop Closing and Map merging in ORBSLAM3

The ORBSLAM3 system not only achieves visual inertial SLAM with very good accuracy, but also provides ORBSLAM-Atlas [13] which enables it with multi-session and multi-mapping capabilities and provides a way to minimize the *hijacked robot* problem. ORBSLAM-Atlas produces a database in which there are a series of maps represented by their keyframes, mapoints, their covisibily graphs and their spanning trees.

By having a dedicated Loop and Map merging thread that queries the Atlas database at keyframe rate for a similar keyframe it enables the entire system to always make use of maps from previous sessions and find potential loop closures.

In case it finds a match for a loop closure it can correct for drift and if the match is from a non-active map the system will do a full bundle adjustment with all keyframes from both the current map and the matched map to create the best possible map for the database. It does this in a separate thread so it does not impair real-time performance.

2.5 Topological Mapping and Semantic Labeling

Topological maps are graph-like spatial representations. Nodes in such a graph often represent states in the agent's state space and edges represent system actions and trajectories that take the agent from a state to another. The meanings of nodes and edges in a topological map vary according to the application as well as the algorithms used to build them. In a mobile robot context, topological map's nodes most often represent a position in space with edges representing trajectories and connections between such positions.

Semantic labeling of images is the categorization of visual information according to an abstract meaning, often related to how humans perceive such information. These meanings often imply specific properties which are useful for the application at hand. In the context of mapping and mobile robots, semantic labeling most often refers to categorizing objects and places.

Topological mapping has been a research topic for many decades with recent advances in computing power and SLAM enabling progress [3] [18]. Whereas semantic labeling is more recent in the literature [1] [7] , with real advances being a result of the foundational work done with machine learning, more specifically, in deep learning and SLAM.

There have been recent works that combine both approaches in a complete system using sensor suites which directly observe depth [1][6] as opposed to ours which does not.

Libraries like [32] present an opportunity for accurate object-recognition which was not possible a decade ago, by making use of neural networks like this, it seems that semantic meaning can be assigned to places that a robot navigates, which in turn could enable a more seamless interaction between humans and robots.

Topological mapping Topological mapping from visual information has been studied extensively in recent years [3][6][18][36].

In order to perform mapping using vision, it is necessary to describe the acquired images and be able to compare these descriptions. Consequently, the quality of the map will directly rely on the method used for visually describing the different environment locations.

The method used for representing the image can be classified into four main categories according to the authors of [18]:

- Methods based on global descriptors, with the image represented by a general descriptor computed using the entire frame as input.
- Methods based on local descriptors, where interest points are found in the image and then

a patch around this point is described in order to identify them in other similar images.

- Methods based on the Bag-of-Words (BoW) algorithm [1], where local features are quantized according to a set of feature models called visual dictionary, representing images as histograms of occurrences of each word in the image;
- Methods based on combined descriptors, where several techniques described above are used together as a new solution.

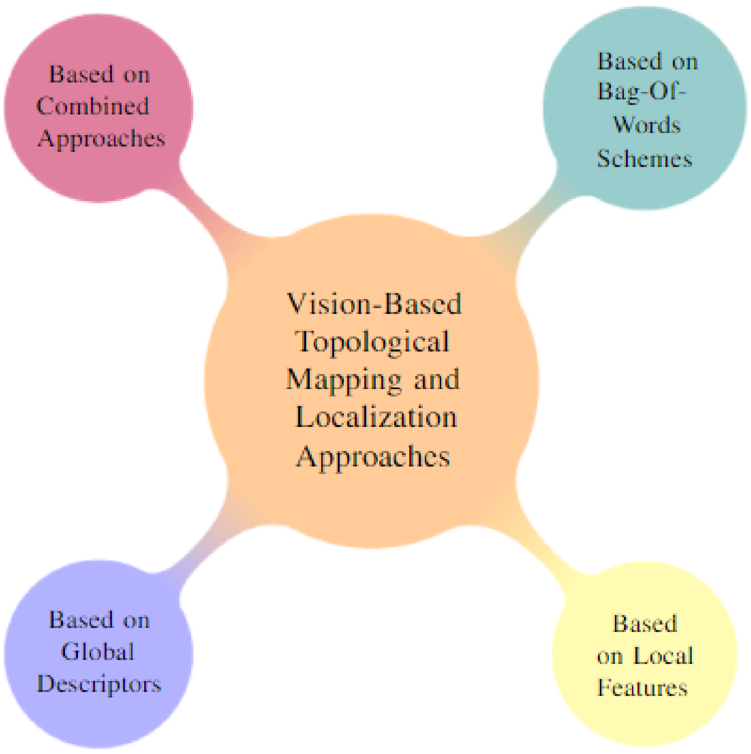

Based on Combined Approaches
Based on Bag-Of-Words Schemes
Vision-Based Topological Mapping and Localization Approaches
Based on Global Descriptors
Based on Local Features

Figure 2.3: Classification of vision-based topological schemes according to their image representation method. From [18]

There are advantages and disadvantages to each of these representation schemes. In table 2.1 a breakdown of these can be seen.

These image representations serve the purpose of allowing for fast and accurate image matching which concerns topological mapping in terms of aggregating similar frames to the same place node.

[1]https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision

Factor	Global Descriptors	Local Features	BoW Schemes
CPU Load	Low	High	Medium
Storage Need	Low	High	Medium
Matching Complexity	Medium	High	Low
Discrimination Power	Low	High	Medium
Perceptual Aliasing Effect	High	Low	Medium
Large-Scale Operation	Medium	Low	High
Spatial Loss Information	Medium	Low	High
Pose recovery complexity	High	Low	Medium

Table 2.1: Advantages and disadvantages for each representation scheme. From [18]

Topology based on geometry A different approach which concerns itself with the geometry of the environment has also been studied, for example in [3]. From the 3D structure of the environment, occupancy information can be retrieved and it is possible to create a set of convex free-space clusters, which form the vertices of the topological map.

This approach lends itself more to sensors that provide depth information such as RGB-D cameras, stereo cameras and LiDAR. However, there have been good results in estimating the geometry of an environment through RGB only video [26] which in turn could result in this approach constructing accurate topological maps using information from a simple RGB camera alone.

In our proposal we follow the geometric approach to topological mapping. However, we do not use a sensor suite that directly estimates distance like the ones mentioned before and thus could decide to pursue a machine learning approach like [26]. Instead, we believe even low accuracy estimation of the 3D structure of the environment, using a monocular camera, can be enough to find geometric features to localize a mobile robot and attempt to create topological maps based on 3D clusters found in the navigated scene.

Chapter 3

Mobile Robot

In this chapter the developed mobile robot will be presented along with the software tools necessary for its operation. The main components are detailed below along with the full system overview.

The system architecture used in this the dissertation work is based on the setup constructed in [27] and [9]. These provide an already working PC interface that can interact with a Raspberry Pi over Wi-Fi to receive the sensor's data and send command signals. They also provide software capable of receiving IMU's readings to an Arduino which then feeds those to the Raspberry Pi.

In section 3.1 a global vision of the system is provided. In particular are detailed the information flows and how data is processed. In section 3.1.1 the software created by Cruz [9] is briefly introduced. In section 3.1.2 all the robot components are detailed and images of the physical robot are provided, as well as all the wiring connections.

3.1 Global System

As depicted in figure 3.1, the hardware can be divided in two main subsystems: the mobile robot and the user's PC. The first one provides the visual-inertial data, while the latter has to be able to receive said data, store it, process it and send control signals for the motor driver back.

While it would be preferable to have all calculations made onboard, with current affordable micro-controllers or micro-computers, the computing power would not be enough for the system to handle real-time performance. Therefore, major computations are offloaded to the user's PC inside Matlab. Communication between subsystems is made via a TCP/IP client-server system created by Cruz [9].

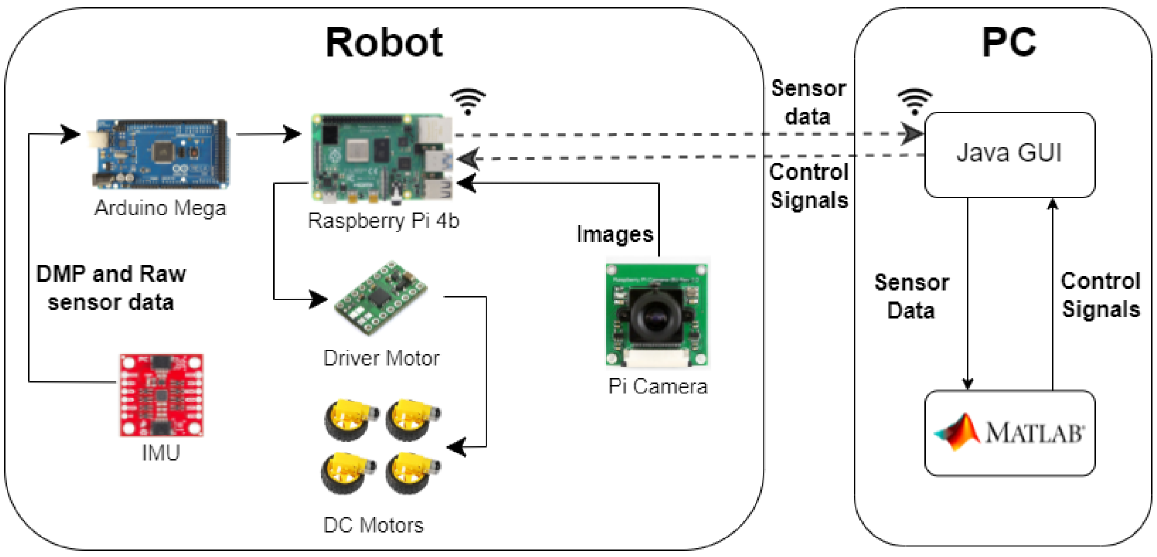


Figure 3.1: General overview of the system

3.1.1 PC Subsystem

The user's PC connects to the mobile robot via a client connection. A Java GUI was written to allow for a better user experience, both for connecting to and controlling the robot. Matlab has to be configured to work with the Java GUI. It is inside Matlab that all computations and processing of data occur. This has several advantages, namely, the easiness of plotting and fast prototyping for new or improved software. Another advantage is that there is a substantial number of libraries available for Matlab which can help accelerate and ease the development of the proposal.

3.1.2 Mobile Platform Subsystem

The robot hardware setup follows a similar structure to [27], with some of the components upgraded to newer versions: a Raspberry Pi model 4b 4GB, an Arduino Mega 2560 R3, a Raspberry Pi Camera (B) Rev 2.0, a SparkFun's 9DoF IMU Breakout - ICM-20948, Pololu's DRV8833 Dual Motor H-Bridge driver, a Multi-Chassis 4WD robot Kit (4DC Motors) with battery holder and 2x PowerBanks, with capacity of 5000 mAh and 6000 mAh.

The main computational unit of this subsystem is the Raspberry Pi 4B. It hosts the TCP/IP server to communicate with the PC which also acquires data from the Arduino. The IMU's data is read using an Arduino sketch written in C++, which is then fed to the previously mentioned server script.

No processing or filtering of the data happens in this subsystem with the exception of the digital motion processing (DMP) that occurs on the IMU chip itself. All wiring connections can be consulted in figure 3.2 and the full robot setup is depicted in figure 3.3.

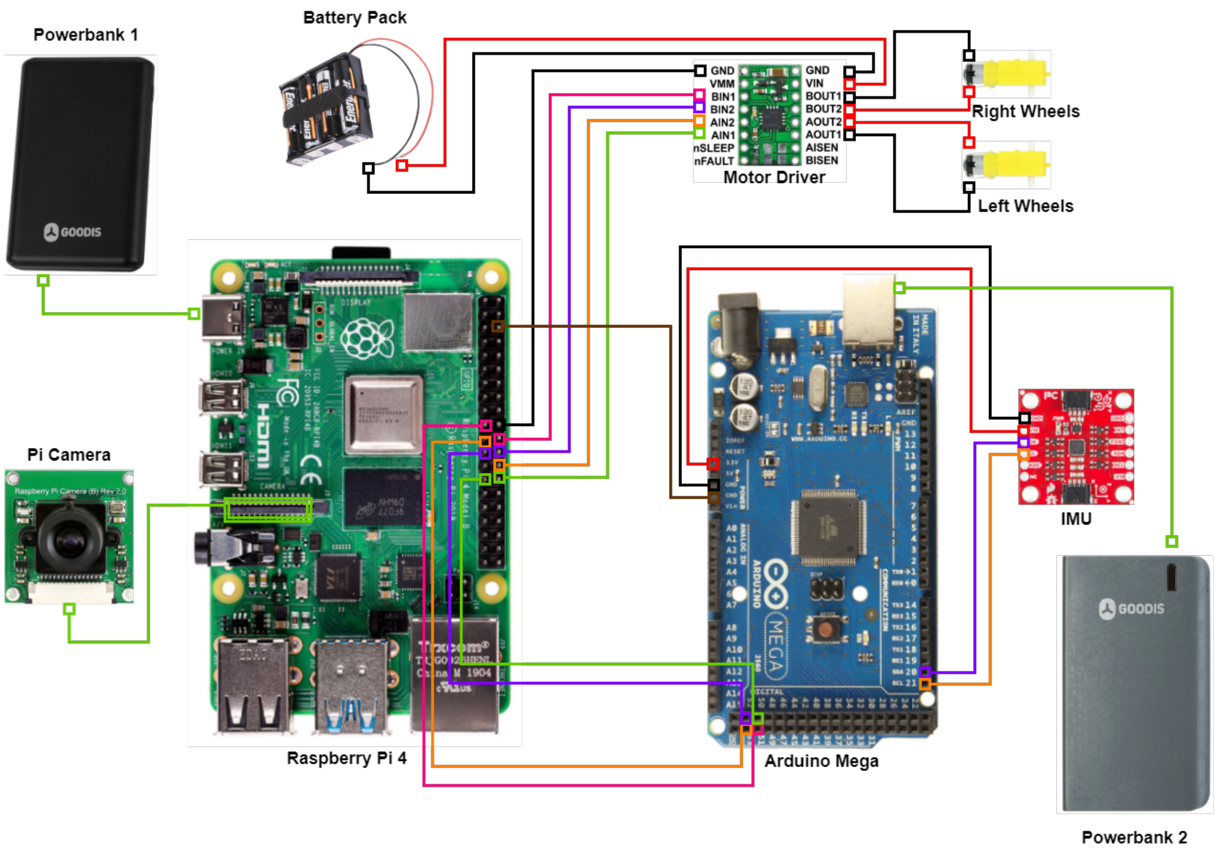


Figure 3.2: Wiring for the mobile system

3.2 Camera Intrinsic and Camera-IMU Extrinsic Calibration

Calibration of sensor systems is important for any task or project where sensors are used. All measurements will have biases or gaussian error that will interfere with system calculations and predictions. In the case of our sensor suite, a camera and an IMU, both intrinsic and extrinsic calibration parameters were estimated. In this section we detail the procedures taken and the calibration results.

3.2.1 Camera Intrinsics Calibration

For calibration purposes we considered the pinhole camera model and the radial-tangential distortion model. To have accurate estimation of landmark positioning using the camera sensor, it is necessary to estimate the camera's intrinsic parameters, the optical center $C = [c_x \; c_y]$, the focal length $f = [f_x \; f_y]$ and the skew coefficient s. With these we can create the camera intrinsic matrix K:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

In addition we also wish to estimate the radial distortion r of the camera by two parameters $r = [r_1 \, r_2]$.

For this purpose, we utilized Kalibr toolbox [1]. It is a ROS toolbox for calibration purposes of various sensors and sensor configurations. It allows for both intrinsic and extrinsic calibration of sensor systems. We used their rolling shutter camera calibration procedure for this calibration [28]. First we recorded a sequence of images with the mobile robot stationary and the calibration target, seen in figure 3.4, being moved in front of the camera as to capture several angles, with the target filling as much of the camera field of view as possible.

This dataset was then converted to a rosbag in order for us to run the calibration library. After calibration the parameter results were:

$$\begin{bmatrix} 1083.4 & 0 & 335.0879 \\ 0 & 1076.4 & 257.0793 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.2}$$

and

$$r = [-0.4373 \; 0.2476] \tag{3.3}$$

[1]Kalibr is an open source library for solving several calibration problems: `https://github.com/ethz-asl/kalibr`

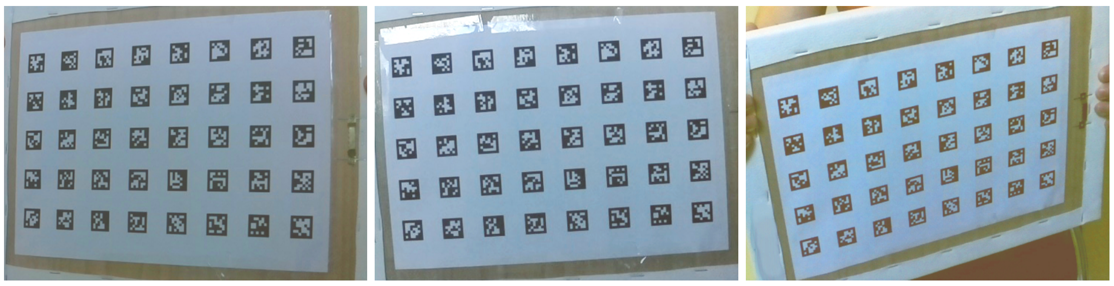


Figure 3.4: Calibration target used for both intrinsic camera parameters and Camera-IMU extrinsic parameters.

3.2.2 Camera-IMU extrinsics Calibration

In order to run our own datasets, we must estimate the rigid trannsformations from the camera to IMU T_{ic}, and from the IMU to the camera T_{ci}.

We once again employed Kalibr toolbox [2] for this estimation. This ROS toolbox enables us to execute a spatial and temporal calibration of a visual inertial system (camera-IMU) [17]. It is essential to note that in order to execute this calibration, a body containing these sensors and ensuring that they remain fixed is required. If the positioning of the sensor changes, a second calibration would be necessary. The prerequisites for this calibration were to supply the toolbox with the intrinsic parameters of the IMU and camera. This includes the noise density and random walk for the accelerometer and gyroscope in the IMU. In an ideal situation, these would be included in the manufacturer's datasheet. The manufacturer for our own IMU did not provide these two values. We used average values found for IMUs of similar quality for both noise density and random walk of each of the two sensors, the accelerometer and gyroscope.

In this calibration, the calibration target is fixed, and the camera-IMU system is moved in front of the target to excite all IMU axes. It is essential to ensure that the calibration target is well-lit and uniformly illuminated, and to keep the camera's shutter speed low to prevent excessive motion blur.

As previously, we converted the acquired dataset to a rosbag in order for us to run the calibration library.

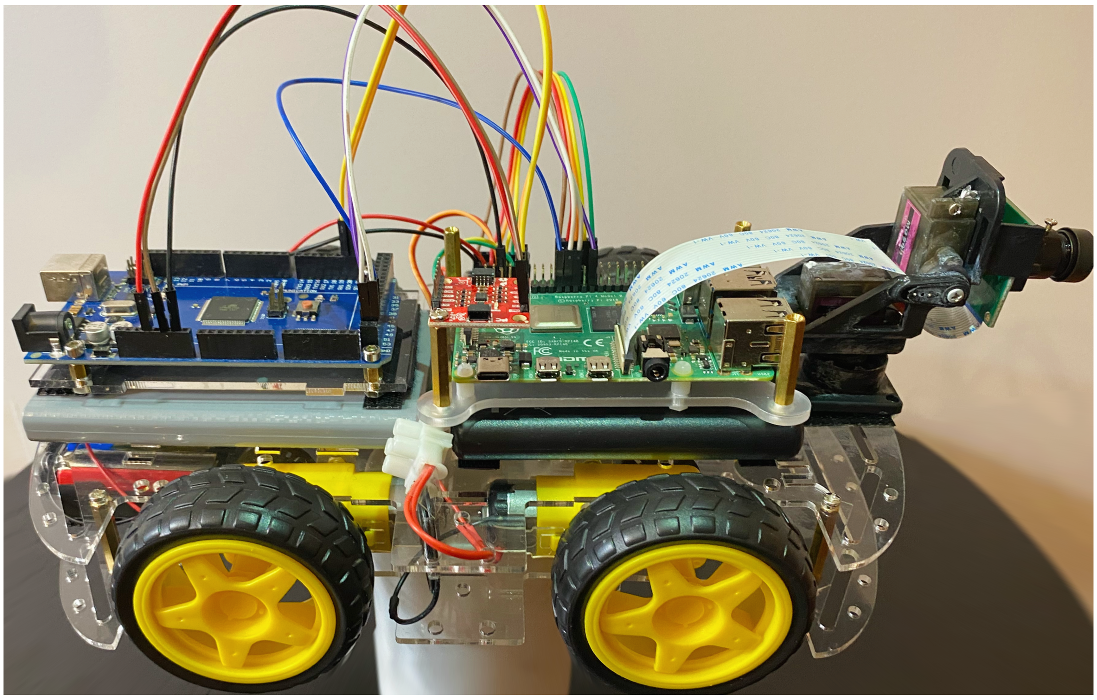
The resulting transformations estimates from the camera to IMU T_{ic}, and from the IMU to the camera T_{ci} are as follows:

[2]Kalibr is an open source library for solving several calibration problems: `https://github.com/ethz-asl/kalibr`

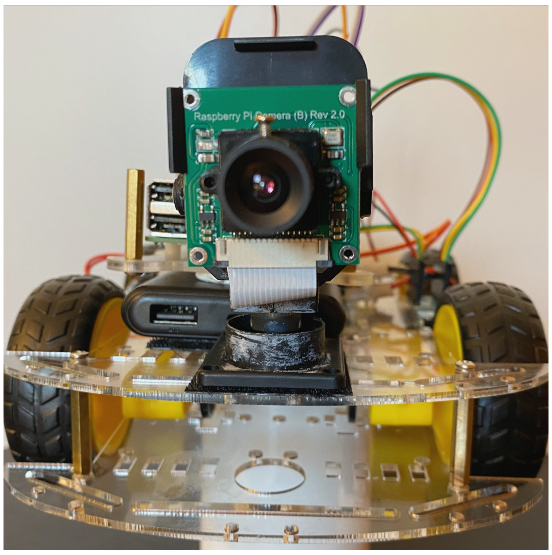
$$T_{ic} = \begin{bmatrix} 0.0116 & 0.0364 & 0.9993 & 0.1903 \\ -0.9993 & 0.0371 & 0.0102 & 0.0396 \\ -0.0367 & -0.9986 & 0.0368 & 0.006 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.4}$$

$$T_{ci} = \begin{bmatrix} 0.0116 & -0.9993 & -0.0367 & 0.0376 \\ 0.0364 & 0.0371 & -0.9986 & -0.0025 \\ 0.9993 & 0.0102 & 0.0368 & -0.1907 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.5}$$

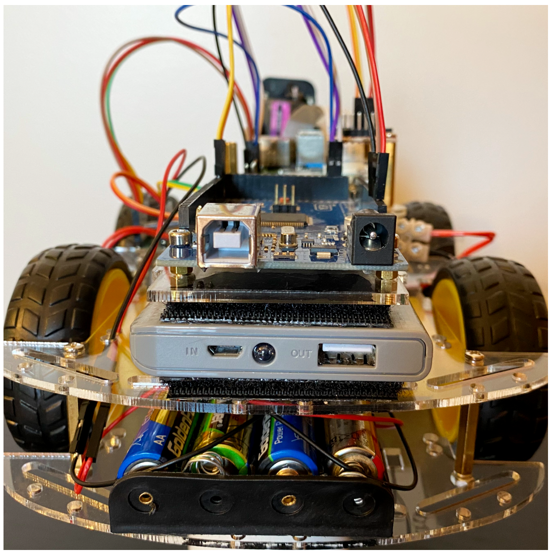
and with temporal shift of the data $[s]t_{imu} = t_{cam} +$ shift estimated at shift $= -0.04175$.



(a) Side view



(b) Front view



(c) Rear view

Figure 3.3: Robot setup. Main components involve a Raspberry-Pi, an Arduino, a Pan-tilt camera, and four DC motors with a dual motor power driver.

Chapter 4

Geometric Scene Labeling and Place Registration

Figure 4.1 illustrates the proposed solution of our work. Given a mobile ground robot doing landmark-based SLAM that loses metric tracking for any reason, is it possible to answer the question "What was the nearest *Place*?".

In order to accomplish this, we generate a topological framework that maps the scene with nodes representing *Places* via a previous navigation that generates a database for comparison.

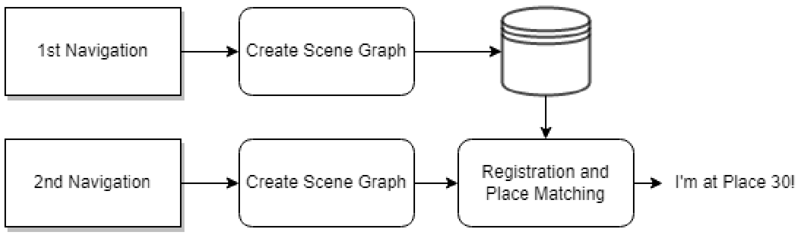


Figure 4.1: Basic diagram of the proposed methodology. Given a sentinel robot navigating a previously mapped environment, can the robot recognize its current location based on the notion of *Places*.

4.1 Hijacked Robot problem

The hijacked robot problem is defined as the ability of the robot to localize itself after being physically moved to a different unknown location. This is a global localization problem for which the robot needs a previous map of the environment to be able to solve it.

The type of solutions used often depend on the SLAM framework chosen. Our proposed methodology is a method that relies on 3D landmark tracking which is one of the most common map representations used for SLAM frameworks. It attempts at providing a solution for this problem without providing a new metric robot pose as output. Instead, it locates the robot topologically, matching its current position to the closest *place*, defined in section 4.3.1.

4.2 Topological Mapping

As defined by Vale [37] topological maps are graph-based representations of environments composed of nodes (or states) and links. Nodes or states can be thought of as separate locations that the robot should be able to identify using external sensors. Links contain information about traversing the connected nodes using internal and/or external sensors and represent paths or actions between endpoints. The current node is defined as the node of the map containing properties that most closely match the measurements from external sensors.

Topological ideas allow for recognition of locations without a need for accurate metric information, providing a potential approach for addressing the hijacked robot problem. In our proposal, location recognition will be based on the overall structure of landmarks tracked by the SLAM system. This structure can be robust to slight metric errors.

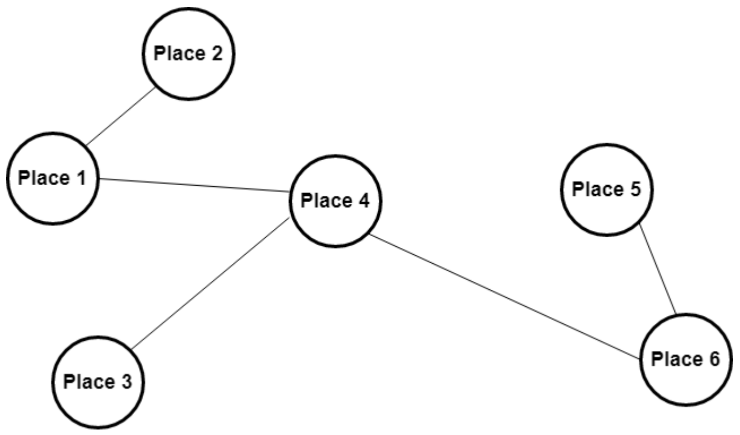


Figure 4.2: Topological graph example. Each node, or state, represents a *Place* with edges between nodes holding information about going from one *Place* to the next. Edges in our proposal hold the bearing angle between the centroids of adjacent *places*.

4.3 Place registration as a solution

As discussed, an advantage of topological mapping is that robot localization does not require highly precise metric estimation and navigation. If the mobile robot system can traverse from node to node and identify at which node it currently is, topological mapping and navigation is a success. By carefully choosing the type of features that identify nodes, topological strategies might help solve the hijacked robot problem.

We propose a mapping strategy that combines information from a metric SLAM system with topological mapping such that to recognize a *place* the robot must only navigate near it and capture it with its camera, as long as the scene has been previously mapped.

The feature type chosen to represent *places* in our topological map is the geometry relation of the 3D landmarks, that are being tracked in the SLAM system. Estimation of structure of the 3D landmarks has robustness to errors in the metric estimation and does not depend on the coordinate system used.

We have built a software system to evaluate the proposal. The system is implemented such that it can use any SLAM framework, provided they output a 3D structure, such as landmarks being tracked.

4.3.1 *Place* definition

The methodology used will be based on *places*. During operation, a SLAM algorithm outputs the 3D landmarks that it is tracking. These 3D landmarks form a sparse pointcloud of the environment. Clusters of landmarks shall be regarded as *places* and will be the basis for recognizing locations in our proposal. *Places* hold the geometric information of every point of the cluster. It holds the cluster's centroid information as well as the images that are associated to each 3D point, if available. The algorithm chosen for the clustering of landmarks is DBSCAN [14].

Places are labeled. In our proposal the labeling method is not explored. In future research labeling might be based on object or room recognition such that *places* have semantic meaning. *Places* are the nodes of the topological graph.

4.3.2 3D pointcloud information

The SLAM system used must provide 3D information of a pointcloud of mapped points. The more accurate the SLAM system chosen is at providing this information, the better the geometric structure of the environment is captured. The chosen SLAM algorithm for providing this information in our work is based on the work of Brossard et al.[25]. We provide both the IMU information and the camera frames, and receive as output the 3D information of the landmarks and the trajectory taken by the robot.

In some experiments we utilize structure from motion software to create the 3D structure of the environment as if it was the output of a SLAM system. This is because initialization is important for the filter's stability and no initialization parameters exist for our own datasets.

4.3.3 Clustering 3D landmarks

DBSCAN for *place* creation

DBSCAN is an acronym that stands for Density-Based Spatial Clustering of Applications with Noise. It is an algorithm for clustering data based on density. The approach aggregates sufficiently dense regions into clusters and identifies clusters of arbitrary structure in noisy spatial databases. It represents a cluster as the most densely connected group of points.

The idea of density-based clustering incorporates the following definitions:

- The neighborhood within a radius ϵ of a certain object is referred to as the object's ϵ-neighborhood.
- If the ϵ-neighborhood of an object contains at least a minimum amount of other objects, *MinPt*s, the object is referred to as a core object.
- An object p is directly density-reachable from an object q if p is within $q's$ ϵ-neighborhood and q is a core object.
- An object p is indirectly density-reachable from object q in a group of objects, D, if there exists a chain of objects p_1,..., p_n, where $p_1 = q$ and $p_n = p$, where p_{i+1} is directly density-reachable from p_i, for $1 \leq i \leq n$.
- An object p is density-linked to another object q in a group of objects, D, if there exists an object o within D from which both p and q are density-reachable.
- Only core objects are densely accessible to one another. This density connectivity relationship of core objects is, therefore, symmetrical. This is not the case for objects other than core objects.
- A density-based cluster is a maximally density-reachable collection of density-connected objects. Each object not contained in a cluster is considered noise.

DBSCAN searches for clusters by examining the ϵ-neighborhood of each datapoint. If the ϵ-neighborhood of datapoint p contains more than $MinPts$, a new cluster is created using point p as the core datapoint. DBSCAN collects directly density-reachable datapoints repeatedly from these core datapoints, which may include the merging of a few density-reachable clusters. When no new points can be added to any cluster, the process removes the corresponding points.

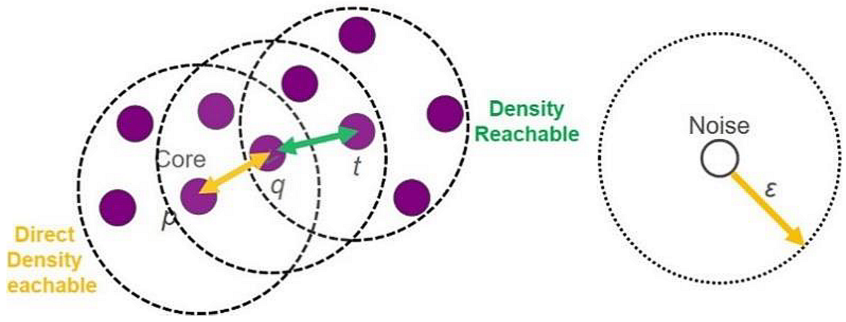


Figure 4.3: DBSCAN visualization. From [20]. In our case, datapoints are 3D points that are clustered by density into *places* that represent nodes in the scene graph.

For our application, the clusters of landmarks found in the first navigation performed will represent *places* and in the second locations which are candidate *places*. The two parameters of the algorithm ϵ and *MinPts* were found experimentally.

Using these clusters of landmarks, we can construct a graph with nodes containing information about a cluster, and edges containing information about the relationship of two clusters. Since navigation of the topological map is not the focus of this work, we chose to relate clusters simply by their relative bearing.

For evaluating our results clusters from the first navigation, *places*, are matched to clusters from the second navigation, *locations*. The quality of cluster matching dictates how well the system behaves.

4.3.4 Graph Split into *Places*

Assignments of names to each cluster is done after *place* creation has happened. The methodology used for naming in our work is simple, using a sequential name assignment. In the future, making use of powerful Machine Learning libraries, it might be possible to conduct analysis of cluster images such that each *Place* may have semantic meaning, eg. "chair", "room corner", "bed" or others.

4.3.5 Registration

In 3D computer vision, registration is one of the most important problems. The objective of registration is to identify the transformation that best aligns multiple collections of points in distinct coordinate systems into a common coordinate system. We consider the registration of pointclouds obtained by a SLAM framework after each of the two navigations. We will use the resulting alignement-transformation to change the robot position from scene2's coordinate system to scene1's such that we can find the closest *place*. The chosen registration algorithm for our application is the matlab's implementation of Generalized-ICP which is based on Segal et al.'s work [35].

We will provide context on the chosen Generalized-ICP algorithm by first introducing the standard ICP algorithm, also called point-to-point ICP. Afterwards, we give an explanation of the chosen Generalized-ICP algorithm which is also known as plane-to-plane ICP.

4.3.6 ICP algorithm

With the goal of achieving precise registration of 3D range images, the Iterative Closest Point (ICP) algorithm was created. It aligns two point clouds by computing the rigid transformation between them in an iterative manner. Over the past decades, numerous variants have been introduced, the majority of which can be categorized as influencing one of the six stages of the ICP algorithm [33]:

- A set of points selected from one or both point clouds;
- Matching these points from one point cloud with samples from the second point cloud;
- Appropriately weighing the corresponding pairs;
- Rejecting certain pairs based on an examination of each pair separately or the entire set of pairs;
- Attributing an error metric to the point pairs;
- Solving the optimization problem.

Due to the fact that all ICP variants may become trapped in a local minimum, they are generally only applicable when the distance between the point clouds is already sufficiently small. This deficiency is frequently addressed by estimating an initial transformation using

Algorithm 1 Standard ICP Algorithm

```
input : Two pointclouds: $A = \{a_i\}, B = \{b_i\}$
        An initial transformation: $T_0$
output: The correct transformation, $T$, which aligns $A$
        and $B$
1  $T \leftarrow T_0$;
2  while not converged do
3      for $i \leftarrow 1$ to $N$ do
4          $m_i \leftarrow \texttt{FindClosestPointInA}(T \cdot b_i)$;
5          if $||m_i - T \cdot b_i|| \leq d_{max}$ then
6              $w_i \leftarrow 1$;
7          else
8              $w_i \leftarrow 0$;
9          end
10     end
11     $T \leftarrow \underset{T}{\text{argmin}} \{\sum_i w_i ||T \cdot b_i - m_i||^2\}$;
12 end
```

methods or algorithms that converge to the global minimum but with lower precision. However, many applications create and process point clouds at a fast enough rate that the precondition of small distance is met.

The simplest algorithm, which is depicted in algorithm 1, is frequently referred to as the Standard ICP [35]. To estimate the transformation T between the model pointcloud A and the data pointcloud B, the algorithm requires the model point cloud A and the data pointcloud B. In addition, an approximation transformation T_0 may be considered. If the unknown, actual transformation between the pointclouds is sufficiently small, the initial transformation can be set to identity.

Each iteration, each of the N data pointcloud points is transformed with the current transformation estimation and matched with its corresponding point from model A, this operation is depicted in line 4 of algorithm 1. If the Euclidean distance between the pairs exceeds $dmax$, matches are rejected by setting their weight to 0, as can be seen from line 5 to 9.

The next iteration begins after solving the optimization problem in line 11, which is to find the transformation T that minimizes the sum of weighted squared distances between the transformed points and their corresponding match in the target pointcloud.

The algorithm converges when the difference between the estimated transformations of two consecutive iterations becomes small or when a predetermined number of iterations is reached.

4.3.7 Generalized ICP

Standard ICP is a point-to-point method, which attempts to align all matched points precisely by minimizing their Euclidean distance. This does not take into account the fact that an exact matching is typically not possible due to the different sampling of the two point clouds, which leads to pairs that do not have perfect equivalence with one another.

Some variants of ICP take advantage of surface normal information to circumvent this issue. In contrast to point-to-point variants, point-to-plane variants take into account the surface normals of the model point cloud. In addition, plane-to-plane variants take into account the surface normals of both the model pointcloud and also the data pointcloud. The Generalized-ICP algorithm introduced by Segal et al. [35] falls under the second classification.

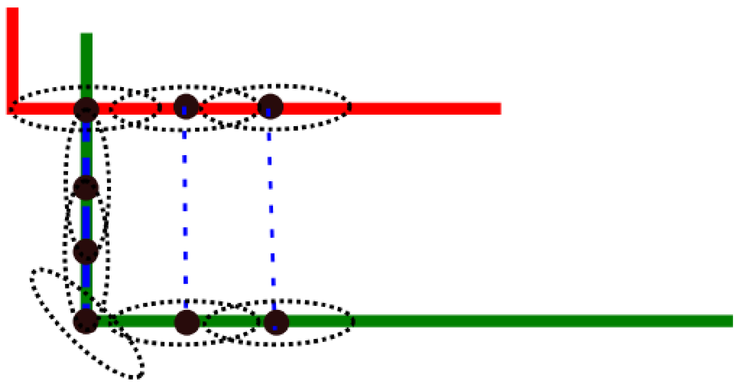


Figure 4.4: Plane to plane matching. From Seagal et al. [35]

By applying a probabilistic model to the problem, it modifies the error function in line 11 of algorithm 1 and assigns a covariance matrix to each point. This is based on the assumption that each measured point corresponds to a real point and that the sampling can be represented by a multivariate normal distribution. This results in the new error function:

$$T \leftarrow \arg\min_T \sum_i d_i^{(T)^\top} (C_i^A + T C_i^B T^\top)^{-1} d_i^{(T)} \tag{4.1}$$

where $d_i^{(T)} = a_i - Tb_i$, C_i^A and C_i^B are covariance matrices (assuming that all points which could not be matched were already removed from A and B and that a_i corresponds with b_i). This formula is similar to the Mahalanobis distance, without the square root.

It is proposed to use the covariance matrix $\left(\begin{smallmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$ for a point with the surface normal $e_1 = (1, 0, 0)^\top$, where ϵ is a small constant that represents the covariance along the normal. This covariance matrix must be rotated for each point based on its surface normal.

They key benefit of introducing information about the surfaces of both pointclouds is that if the matching of points reveals inconsistent surface matching, those correspondences will contribute less in the optimization problem. This improves the registration process and provides more robustness to incorrect correspondences in most indoor scenarios since human constructions are highly structured. This is preferred for our approach since we aim to construct topological maps of indoor home environments.

4.4 Software diagram and Summary of proposal

We have built a software system to evaluate the proposal. The system is implemented such that it can use any SLAM framework, provided they output 3D structure, such as landmarks being tracked.

We have used an unscented kalman filter on Lie groups for fusion of IMU measurements and camera frames based on Barrau et al. [25]. Since the work by Barrau et al. [25] takes advantage of IMU measurements we focus on the output of this SLAM framework for some experiments.

The methodology it follows is:

1. Begin first navigation and get information of the scene from SLAM system and all the landmarks as a pointcloud.
2. Create clusters of landmarks using DBSCAN [14] clustering method on their 3D position.
3. Assign *places* label to each landmark having each landmark belonging to the same cluster considered the same *place*.
4. Begin second navigation and get information of the scene from SLAM system and all the landmarks as a pointcloud.
5. Match the second navigation's clusters landmark pointcloud to the prior clusters pointcloud based on plane-to-plane ICP [35].
6. Use the transformation obtained to transform robot's position, and *places* centroids from second navigation to first.
7. Compute euclidean distance to all *places* and localize the robot as being at the nearest *place*.

Using this methodology a graph scene composed of *places* as states is created and matched to a previous state. The amount of *places* created is dependent on the 3D observed scene at the time of clustering and as such may be different from the first navigation to the second. We assume that the structure of the clusters created is similar enough that a good matching can be created between navigations.

To evaluate each experiment success, we consider images from the matched clusters using the process depicted in figure 4.5. We plot the images that generated the 3D points of a *place* from Navigation1 and its correspondent from Navigation2. Since the amount of images per cluster is too large for it to be feasible to show all images, we plot both a reference image for the *place* and the mean image. The representative image i_r of *place* X with images i and mean image i_m was chosen as such:

$$i_r = \min_{\forall i \in X} |i - i_m| \quad . \tag{4.2}$$

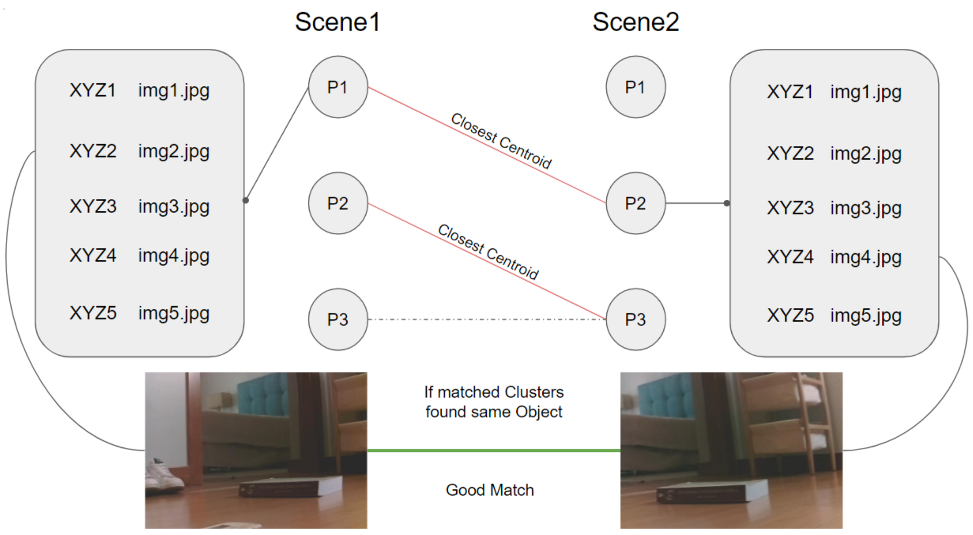


Figure 4.5: To evaluate matching success we employ the strategy depicted above. Each *place* holds information about each 3D point it contains as well as the associated image from which the 3D point was first generated. We compare *place* matches by comparing a representative image of each matched cluster and the average image of each matched cluster.

A complete diagram of the implemented software design can be seen in figure 4.6. It illustrates each of the main steps implemented characterized as a rounded block. Each of the main blocks is independent of the other ones provided the data structures are kept compatible. This leaves room for use of a different SLAM system for example. Future study may potentially also consider a more advanced form of graph split labeling in which pictures and machine learning are used to classify clusters as real-world objects.

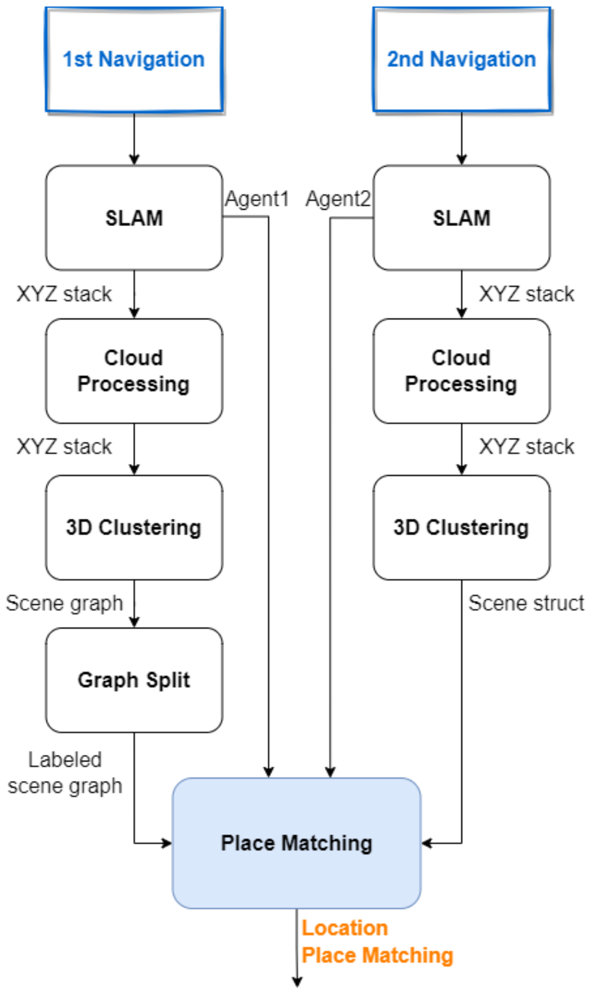


Figure 4.6: Diagram of complete software implementation

Chapter 5

Experiments

In this section we will present three types of experiments that were conducted to evaluate the *place* registration methodology for solving topological localization recognition.

For each type of experiment, pairs of robot navigations are performed. The first represents the scene acquisition and *place* generation for topological mapping. The second represents scene navigation from which we can verify if current *place* recognition was successful.

We will first present the different experiment setups. After that we present the obtained results, followed by an analysis of the data and a discussion of the experiment's success. To conclude a critical analysis of which areas can introduce the most improvement for future research is suggested.

We utilize the public EuRoC MAV Dataset Vicon Room 1 02[4], for one of the experiment setups since the authors of [25] provided initialization parameters specific to this dataset. For the experiment using this dataset we utilized the SLAM filter [25] described in section 2.3.

All other datasets were acquired with the mobile robot described in chapter 3.

Since achieving SLAM is not the focus of our work, we utilize a structure from motion software package, VisualSFM [1], to acquire a 3D pointcloud for home datasets. We then apply our proposed methodology on this 3D structure as if it was the output of a SLAM system.

5.1 Experimental Setups and Implementation Details

All types of experiments conform to the same fundamental structure.

First, a navigation is performed to map the surroundings and collect the labeled scene graph

[1]VisualSFM is a GUI application for 3D reconstruction using structure from motion (SFM).
`http://ccwu.me/vsfm/index.html`

and model pointcloud. This navigation will be referred to as Navigation1.

Second, another navigation is carried out, this time with the conditions being altered based on what type of experiment is being carried out. This navigation will be referred to as Navigation2. It will observe the same or a portion of the same scene as Navigation1 and produce a new scene graph and pointcloud that will be matched to the first one.

Setup I For experiment setup I, we will present results with EuRoC MAV Dataset Vicon Room 1 02[4].

For this experiment we will analyse the simplest case.

Navigation2 will be a subset of the dataset used for Navigation1. Since Navigation2 is a subset of Navigation1, the observed scene is also a subset of the previous scene with precisely the same perspective of observation.

This experiment serves as a proof of concept for the methodology and, in case of failure, would demonstrate a fundamental problem with either the approach or the sub components.

This is the only setup for which we use the public Euroc dataset.

Setup II All other datasets are acquired at the house of the author. Common house objects, such as books, were spread out through the scene so that the SLAM system and the structure from motion software have more visual features to track.

The sentinel mobile robot navigates through a room at a slow speed, with some stops and with several turns in its trajectory. It acquires visual and inertial data from its sensors which is sent to the PC software system to perform all calculations. It performs one of two trajectories that were traced on the ground using coloured tape.

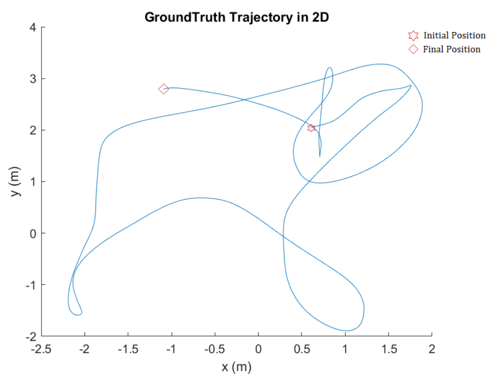
For this experiment Navigation1 and Navigation2 observe the same scene with very similar trajectories, Navigation2 being a re-acquisition of Navigation1. No modifications were made to the environment. The second navigation starts and ends at a different, but similar, location to the first navigation.

Setup III For this setup, the robot takes a different trajectory observing the same scene. The trajectory of the mobile robot in Navigation1 is the same as for Setup II Navigation1. For Navigation2 a new dataset using a different trajectory was acquired. A estimation of the groudtruth for both navigation can be seen in figure 5.9.

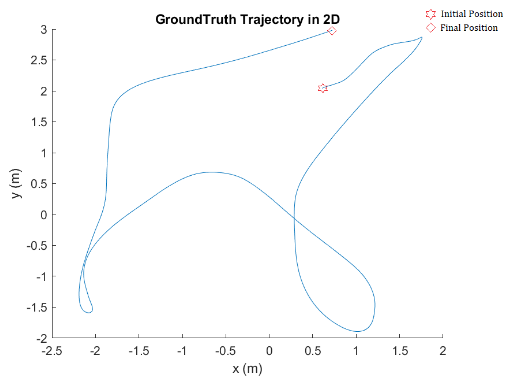
This is a more complex case than the previous experiments. With different perspectives on the scene and a different trajectory, it is more challenging to obtain a 3D structure that is comparable between navigations. We postulate that clusters will still form according to the objects seen, permitting correct matches.

5.2 Experiment I: EuRoC Datasets

This experiment uses Setup I. The navigations used for this are taken from the EuRoC MAV Dataset Vicon Room 1 02 dataset. We use the SLAM system from Barrau et al. [25] and utilize this public dataset since it is the only one for which the chosen SLAM system provides the initialization parameters for. We use this SLAM filter to compute both the trajectory of the robot and the 3D structure of the scene. We then use this output and perform our proposed methodology of *place* clustering and *place* matching.



(a) Ground truth path for navigation 1



(b) Ground truth path for navigation 2

(c) Nav.1 frame 1

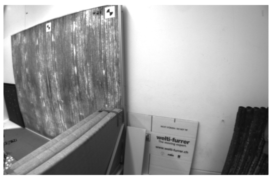
(d) Nav.1 frame 79

(e) Nav.1 frame 141

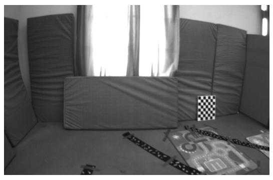
(f) Nav.1 frame 368

(g) Nav.2 frame 1

(h) Nav.2 frame 135



(i) Nav.2 frame 214



(j) Nav.2 frame 414

Figure 5.1: Representation of Navigation1 and Navigation2 for Experiment I. From left to right, first frame, early middle frame, late middle frame, last frame.

Navigation1 initiates at the same time as Navigation2. Navigation1 composes 6000 IMU iterations and 600 frames. Navigation2 composes 4000 IMU datapoints and 400 frames. A representation of Navigation1 and Navigation2 can be seen in 5.1.

From running the Barrau et al.[25] software we receive as output both the trajectory of the mobile robot and the 3D structure of the environment. The filter only holds thirty landmarks at all times, removing old ones when a new 3D landmark is required. We require more that thirty 3D points to evaluate our proposal and so we externally save all landmarks the first time they are added to the filter. This means they are never updated inside the filter after we save them which, in turn, means there is less accuracy.

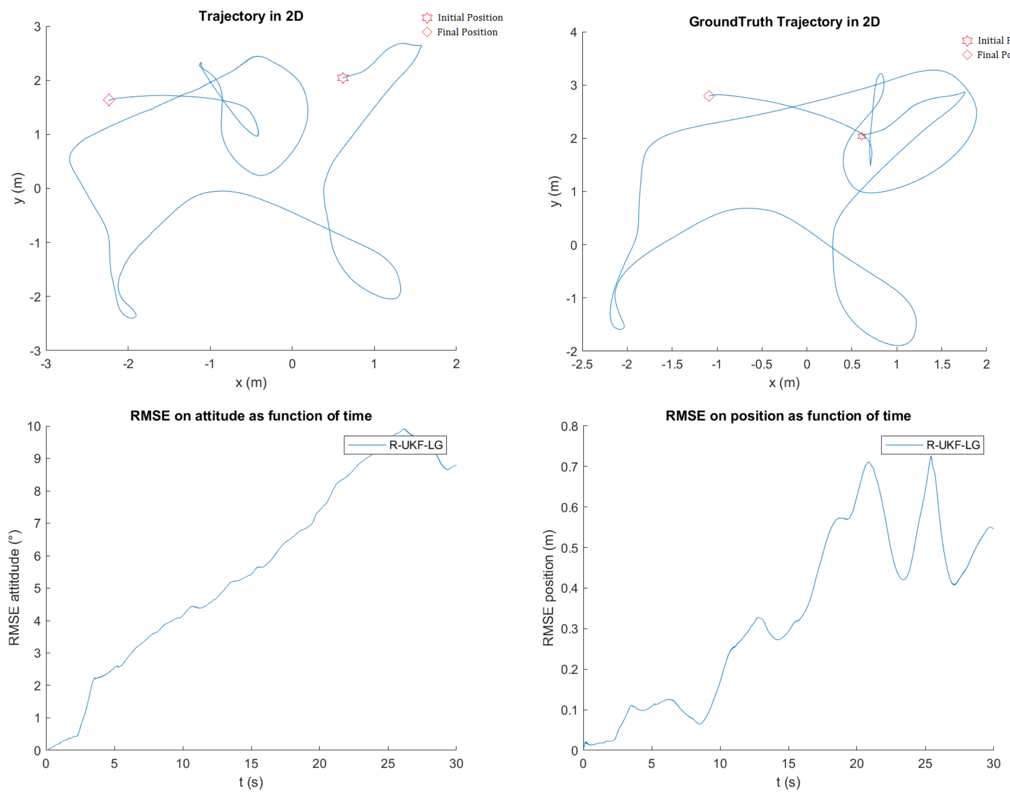


Figure 5.2: Filter estimation results for Navigation1 for orientation error and trajectory error.

Even with less 3D geometric accuracy we postulate that as long as there can be a match between scenes, clusters will form around the same locations and *places* can be extracted and matched.

After running the SLAM filter we obtain the 3D pointclouds and the robot trajectory, as well as the comparison with groundtruth. We show in figure 5.2 and in figure 5.3 the filter estimations and their groundtruth comparison for Navigation1 and Navigation2 respectively. The filter can at first very accurately track position and orientation of the robot with errors accumulating as more iterations are run. These tracking results should provide quality 3D structure estimation

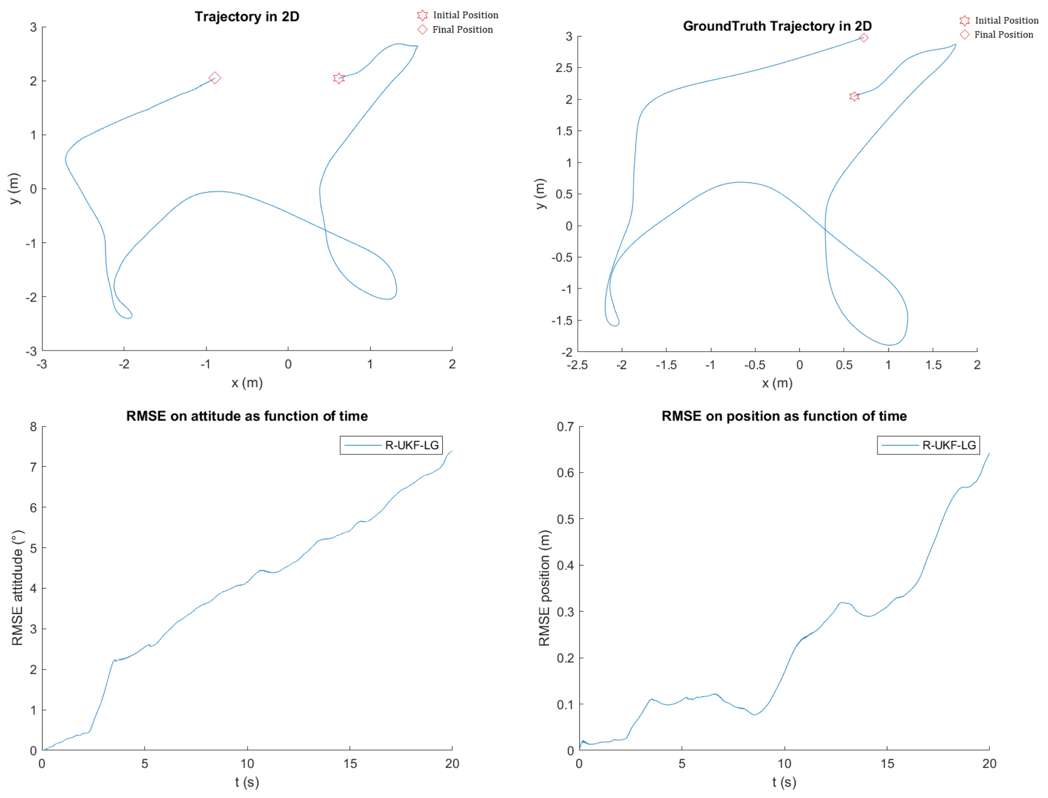


Figure 5.3: Filter estimation results for Navigation1 for orientation error and trajectory error.

that is enough to attempt *place* construction and scene matching.

In figure A.1 we show the pointclouds without normalization and no centering. The dimensions used are the dimensions estimated inside the filter. The pointclouds produced for Navigation1 and Navigation2 were identical up to the point of Navigation2 cutoff. This is expected since the navigations utilize the same dataset, with Navigation2 using a subset of Navigation1. Scene matching is thus significantly easier in this experiment in terms of the structure of the scene.

For registration using plane-to-plane ICP we utilized an inlier-ratio of 0.75 for this pair of navigations. The registration from scene1 to scene2 shows near perfect results in terms of visual appraisal. The root average squared error of the registration was 0.0023 m. It is understandable that registration is done very successfully since we utilize a subset of the same dataset in Navigation2 compared to Navigation1 and the filter created an identical pointcloud up to the cutoff point of Navigation2.

In figure A.2 we show the pointclouds from an above view with the coloured clusters obtained from DBSCAN, note the axis' relative size is not kept. The clustering parameters, cluster distance and minimum points per cluster, used for this experiment were 0.3 m and 11 respectively for both navigations. For this experiment the amount of clusters created in Navigation1 was ten and for Navigation2 was six. Between navigations we can observe that there are some clusters that Navigation1 could produce due to more data points that Navigation2 could not.

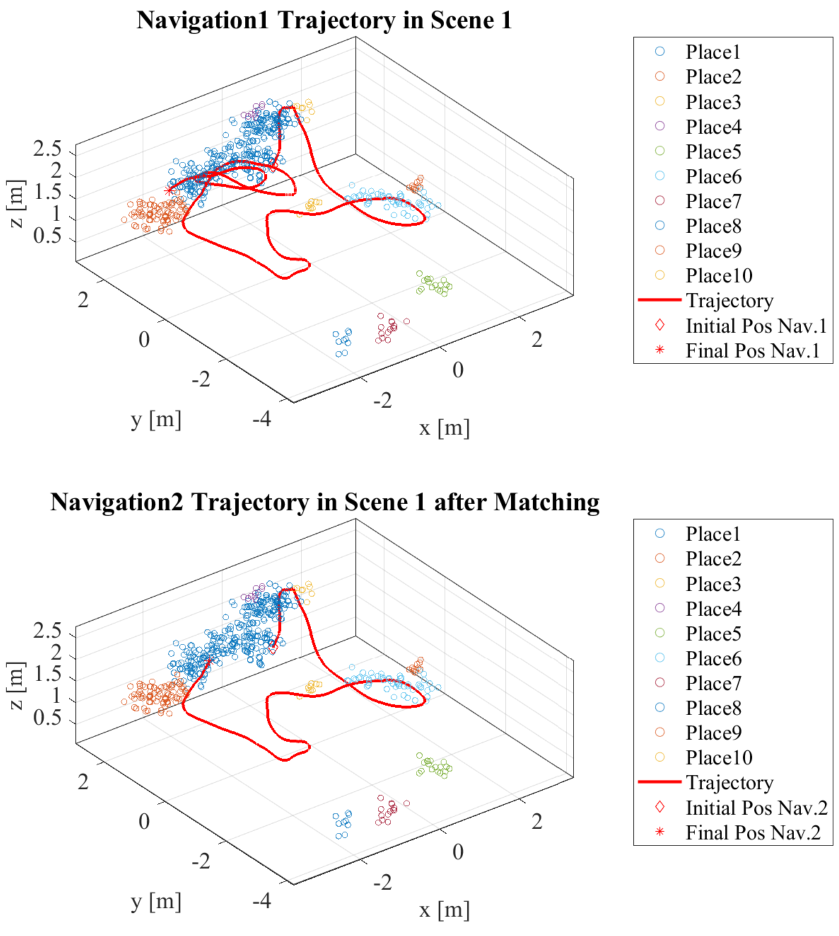


Figure 5.4: Estimated trajectory of Navigation1 in scene1 (left) and estimated trajectory of Navigation2 in scene1 after matching it to the correct frame of reference. In both plots we show the found *places* from Navigation1.

Another interesting observation is that some clusters in Navigation1 further divided into sub-clusters compared to Navigation2. Visual inspection suggests that clusters have been created in the same general vicinity of the scene, further analysis of this is done when we present the matching results.

Matching Results - From the previously shown alignment of scenes we have available the rigid transformation from scene2 to scene1. By applying the registration to the clusters and matching them using smallest euclidean distance we obtain the matching shown in figure 5.5.

To evaluate how good the matching process was, we show in figure A.3 a representative image for each *place* and the average image associated to the *place*, with the standard brightness deviation shown below.

Since images are taken in motion and a specific place is observed from several close but different perspectives it is normal for average images to present some blur. Because this dataset is from a drone that moves at a relatively high velocity, it is expected that this will happen.

In *Place1* we note that the average image has a very high standard deviation and appears very blurred. This seems to be due to the fact that the cluster associated a lot of 3D points and their related image. While a lower clustering distance might have solved this problem, some *places* could become unrecognizable by the system due to having their core points farther away.

Another observation is that there seemed to be some repetitions of matches. This is because different clusters formed around the same visual area. This could be solved by instead of lowering, increasing the clustering distance such that partitioning of clusters near each other does not happen.

From this we can gather that there is a clear trade off in terms of clustering distance depending on the problems we wish to avoid.

Despite these problems, all place matches have been made correctly, with representative images depicting close locations of the room.

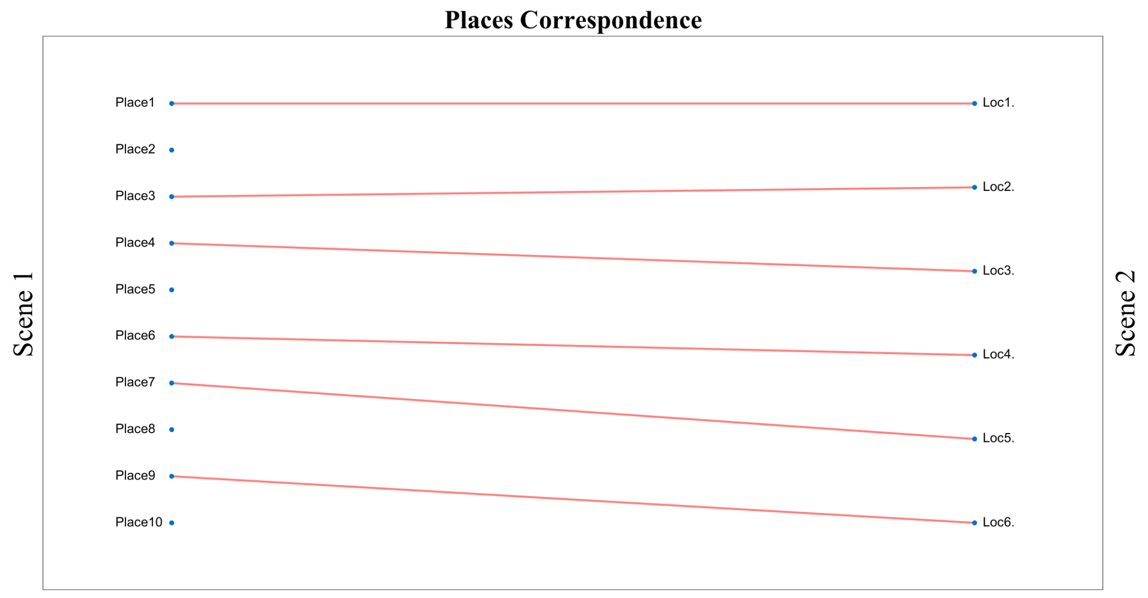
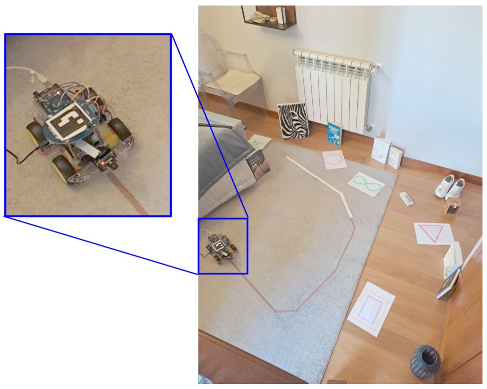
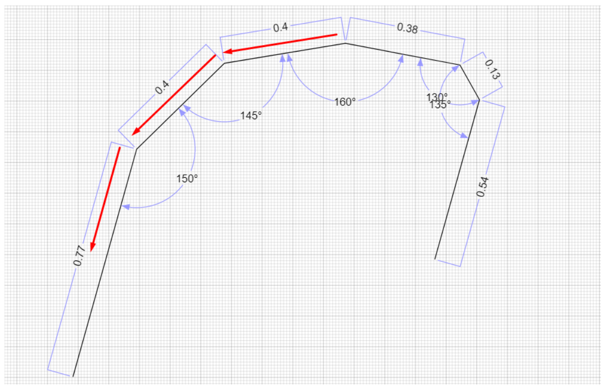


Figure 5.5: Visual representation of *place* matching between navigations, according to closest *place* centroids. For this experiment all clusters found in Navigation2 found a *place* match in Navigation1. However, not all *places* from Navigation1 had a correspondence. This might happen if a different amount of clusters is found or if the closest *place* from Navigation1 is the same for two clusters found in Navigation2, in which case only the closest cluster of Navigation2 is matched to the *place* of Navigation1. Since Navigation1 was an extension of Navigation2 it is reasonable that there are *places* that were not sufficiently visited in Navigation2.

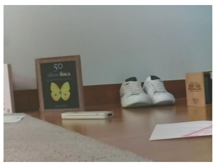
5.3 Experiment II: Similar Paths - Wheeled Mobile Robot

The navigations used for this experiment are a pair of navigations taken at home using the trajectory depicted in red in figure 5.6(b). This experiment uses setup II, with Navigation2 having a similar trajectory to Navigation1 but different starting point and ending point and thus observing the scene from a different but very similar perspective. A representation of Navigation1 and a representation of Navigation2 can be seen in figure 5.6.

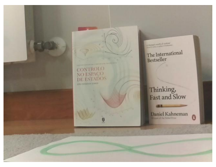

(a) Setup



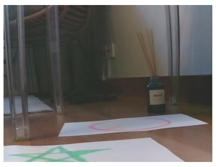
(b) Ground truth path


(c) Nav1 fr.1

(d) Nav1 fr.85


(e) Nav1 fr.164

(f) Nav1 fr.226


(g) Nav1 fr.455

(h) Nav2 fr.1

(i) Nav2 fr.33

(j) Nav2 fr.16

(f) Nav2 fr.247

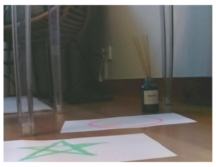

(k) Nav2 fr.447

Figure 5.6: Setup for experiment 2. The photo on the left portrays the scene the robot navigates in. The trajectory portrayed on the right is in [cm]. The red arrows indicate the sections of the trajectory taken by the robot for this navigation. We preconstructed the trajectory by using colored tape on the ground and then took measurements, using measuring tape and a protractor, for the length of each section and the angle between them respectively.

In this experiment, in order to avoid the precise calibration of camera and IMU sensors, required in [25], and the initial landmarks initialization, we produced the 3D structure of the scene by using a structure from motion software, VisualSFM [2]. We use the pointcloud of this

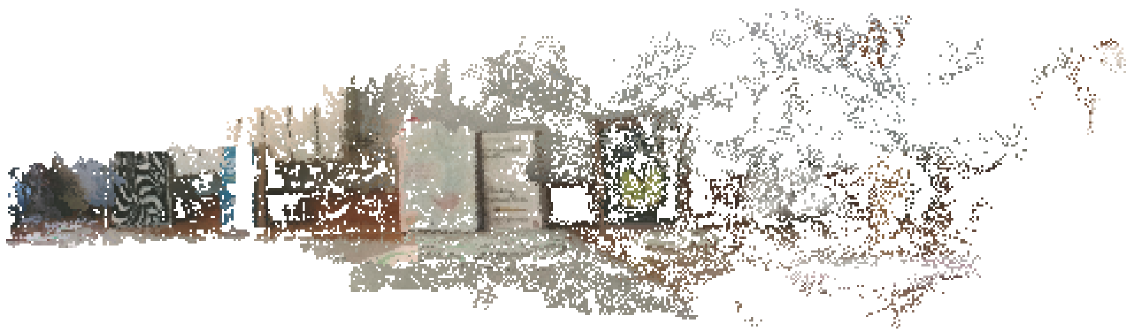
[2]VisualSFM is a GUI application for 3D reconstruction using structure from motion (SFM). `http://ccwu.me/vsfm/index.html`

software as if it was the output of a SLAM system, and associate *places* from navigation to navigation. The localization module is, however, turned off since there is no trajectory or pose estimation output. Using just images implies a scale reconstruction ambiguity. Because of this, Navigation1 and Navigation2 pointclouds are not constructed at the same scale. To deal with this problem we normalize and center both pointclouds.

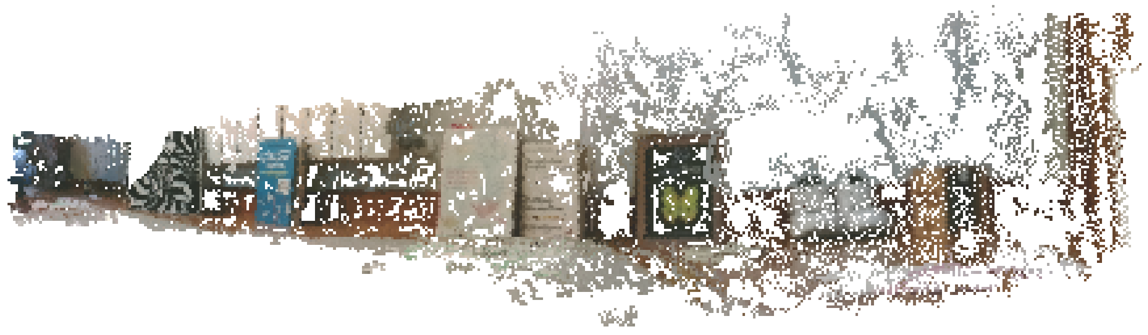
In figure 5.7(c) we show the pointclouds after normalization and centered at the origin. The poinclouds produced for Navigation1 and Navigation2 were fairly similar with some noticeable differences in density in some areas, and different edge points. This is to expect since the navigations did not start and end at the exact same place even if close. The navigations also did not have the exact same trajectory since Navigation2 is a re-acquisition of Navigation1, leading to natural differences in robot motion which is not controlled for.

For registration using plane-to-plane ICP we utilized an inlier-ratio of 0.8 for this pair of navigations. The registration from scene1 to scene2 produced acceptable results in terms of visual appraisal. The root average squared error of the registration was 0.0083 in normalized distance units. For better visual inspection we provide coloured pointclouds for the result of both navigations in figure 5.7(a) and (b).

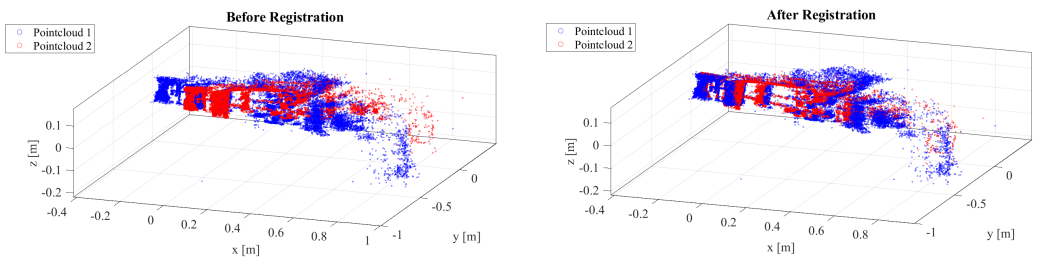
In figure A.4 we show the pointclouds from a front view with the coloured clusters obtained from DBSCAN, note the axis' relative size is not kept. The clustering parameters, cluster distance and minimum points per cluster, used for this experiment were 0.015 of normalized distance and 200 respectively for both navigations. We can confirm from visual inspection that clusters are highly dependent on the 3D structure of the scene. The amount of clusters created in Navigation1 was seven and for Navigation2 was six. The locations of the clusters visually align from one 3D structure to the other which is promising.



(a) Coloured pointcloud obtained from Navigation1



(b) Coloured pointcloud obtained from Navigation2



(c) Pointclouds obtained from Navigation1 (blue) and from Navigation2 (red).

Figure 5.7: Experiment II pointclouds. In (a) and (b) we can observe that the objects in the middle of the trajectory were better captured by VisualSFM, compared to the edges of the scene. Particularly the right side of the scene was not very well captured. A possible reason might be less features for matching in that area, along with having less available frames observing the location. On the left of (c) we show the superimposed pointclouds after normalization and centroid centering without registration. On the right of (c) we show the pointclouds after registration using plane-to-plane ICP with an inlier threshold of 0.8.

Matching Results - From the previously shown alignment of scenes, we have available the rigid transformation from scene2 to scene1. By applying the registration to the clusters and matching them using smallest euclidean distance we obtain the matching shown in figure 5.8.

To evaluate how good the matching process was, we show in figure A.5 a representative image for each *place* and the average image associated to the *place*, with the standard brightness deviation shown below.

Since images are taken in motion and a specific place is observed from several close but different perspectives it is normal for average images to present some blur. But in addition to this, the average images of each cluster show that the range of images associated with clusters includes outliers. This may be due to two main reasons: there were 3D points wrongly clustered and thus their associated image should not have been included, or the 3D point itself should not have been placed there and is a bad estimate from VisualSFM.

Another important observation is the fact that different clusters represent the same real world *place*, being repeated. This is because the 3D points associated with the repeated clusters were observed from the same camera positions. This presents a trade off problem, since if we wish to eliminate repetitions of real world *places* we should increase either the clustering distance or the minimum amount of points per cluster, at the risk of not capturing certain locations as separate *places* and of including more outliers per *place*.

However, both the average images and the representative images of each match show that matching was successful by matching the same general location correctly from Navigation2 to Navigation1.

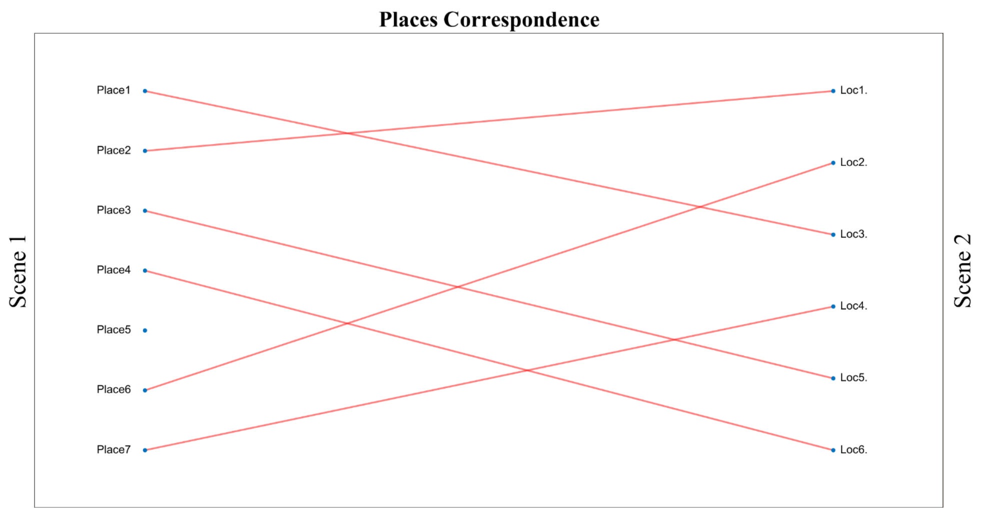
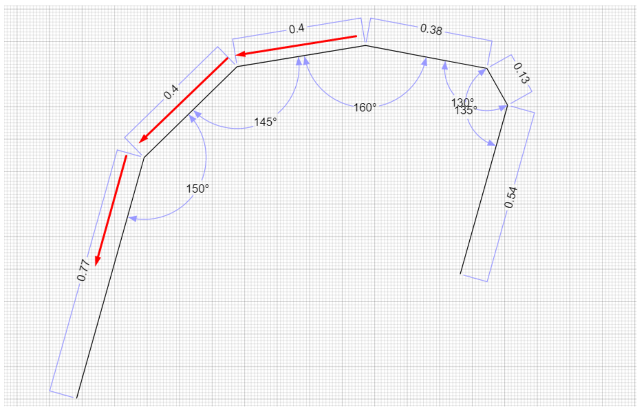


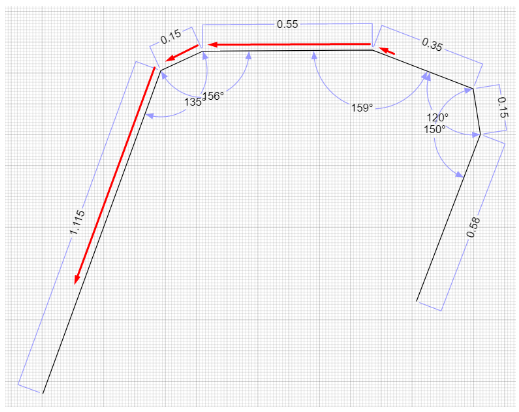
Figure 5.8: Visual representation of *place* matching between navigations, according to closest *place* centroids. For this experiment all clusters found in Navigation2 found a *place* match in Navigation1. However, not all *places* from Navigation1 had a correspondence. This might happen if a different amount of clusters is found or if the closest *place* from Navigation1 is the same for two clusters found in Navigation2, in which case only the closest cluster of Navigation2 is matched to the *place* of Navigation1.

5.4 Experiment III: Different Paths - Wheeled Mobile Robot

The navigations used for this experiment are a pair of navigations collected at the house of the author using approximately the trajectories depicted in red in figure 5.9. This experiment uses setup III, with Navigation2 having a different trajectory to Navigation1 observing the scene from a different perspective. Navigation1 is the same for experiment III as in experiment II.

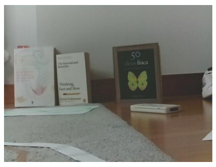


(a) Ground truth path Navigation1



(b) Ground truth path Navigation2

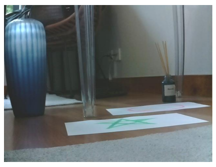
(c) Nav2 fr.1



(d) Nav2 fr.96

(e) Nav2 fr.213

(f) Nav2 fr.286



(g) Nav2 fr.466

Figure 5.9: Setup for experiment 3. The trajectories portrayed are in [cm]. The red arrows indicate the sections of the trajectory taken by the robot for each navigation. We preconstructed the trajectories by using colored tape on the ground and then took measurements, using measuring tape and a protractor, for the length of each section and the angle between them respectively.

As in Experiment II, we utilise VisualSFM [3] to provide us with the 3D structure of the scene for this experiment. We again treat the output of this software as if it was the output of a SLAM system, and associate *places* from navigation to navigation. The localization module is, once again, turned off since there is no trajectory or pose estimation output.

In figure 5.10(c) we show the pointclouds after normalization and centered at the origin. The poinclouds produced for Navigation1 and Navigation2 were fairly similar with some noticeable differences in density in some areas, and different edge points. This is expected since the navigations took a different trajectory and did not start or at the same place even if close.

[3]VisualSFM is a GUI application for 3D reconstruction using structure from motion (SFM). `http://ccwu.me/vsfm/index.html`

For registration using plane-to-plane ICP we utilized an inlier-ratio of 0.45 for this pair of navigations. The root average squared error of the registration was 0.0057 in normalized distance units. For better visual inspection we provide coloured pointclouds for the result of both navigations in figure 5.10(a) and (b).

In figure A.6 we show the pointclouds from a front view with the coloured clusters obtained from DBSCAN, note the axis relative size is not kept. The clustering parameters, cluster distance and minimum points per cluster, used for this experiment were 0.025 of normalized distance and 250 respectively for both navigations. The amount of clusters created in Navigation1 was seven and for Navigation2 was six.

Since the scene was observed from a different perspective, we can notice a change in the structure of the resulting pointclouds and therefore, the most dense regions. Regions that were very well represented in Navigation1, such as the zebra on the left side of the scene, were not as well represented in Navigation2. In both navigations the right side of the scene has lower density of points. This is to be expected since fewer objects were present there. There were also fewer frames modeling the right area. This phenomenon can present a problem for the methodology if strong enough. However, as we will present in the matching section, for this experiment the results were still comparable to previous experiments.

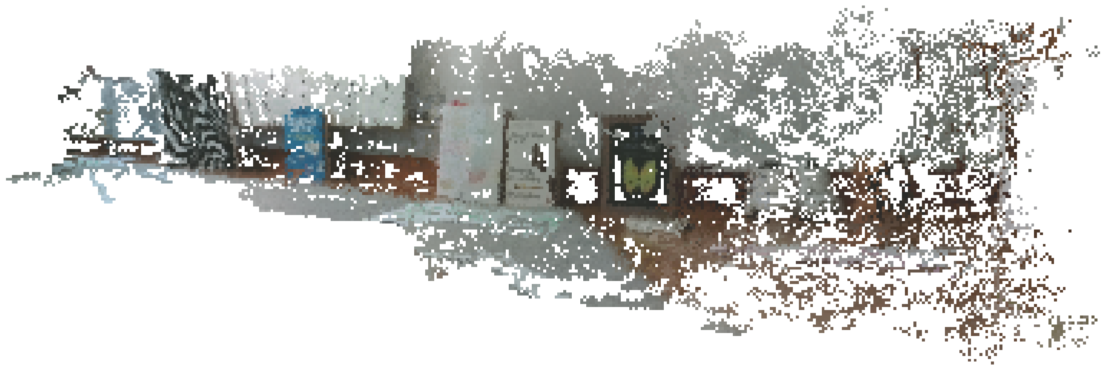
Matching Results - From the previously shown alignment of scenes we have available the rigid transformation from scene2 to scene1. By applying the registration to the clusters and matching them using smallest euclidean distance we obtain the matching shown in figure 5.11.

To evaluate how good the matching process was, we show in figure A.7 a representative image for each *place* and the average image associated to the *place*, with the standard brightness deviation shown below.

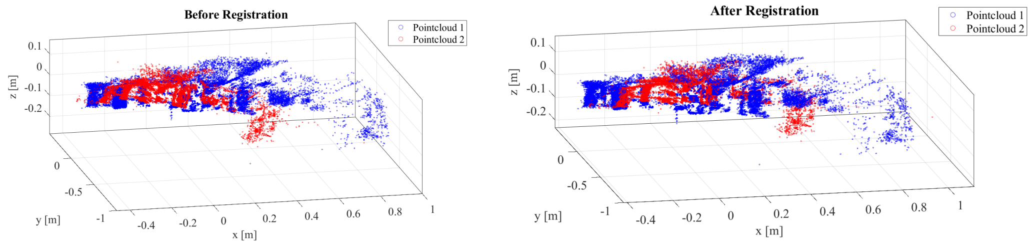
Results for this experiment seem to match the previous results. Blurring of average cluster images can still be observed. As mentioned before, this is likely due to two reasons. The use of images during motion, which displace the view slightly. And the inclusion of 3D points in the cluster that are farther off from the average image. This can be tuned by considering lower values for clustering distance but might lead to inability to model some *places*. For this experiment, remarkably, less outliers can be seen in average images. This may me due to a better choice of parameters, or the structure of the pointcloud resulting from VisualSFM having modeled real world objects better.

We can still observe some overlap of *places* happening, however, both the average images and the representative images of each match show that matching was successful by matching the same general location correctly from Navigation2 to Navigation1.

(a) Coloured pointcloud from Navigation1
(same as Fig. 5.7(a), repeated for comparison)



(b) Coloured pointcloud from Navigation2



(c) Pointclouds obtained from Navigation1 (blue) and from Navigation2 (red).

Figure 5.10: Experiment III pointclouds. In (b) we can observe that contrarily to Navigation1 of this experiment and Navigation2 of Experiment II, the left side near the end of the trajectory of the robot was not as well modeled. The middle and right side, however, have better modeling. This could lead to better *place* matching in the areas where both navigations had good intersection and worse in others.

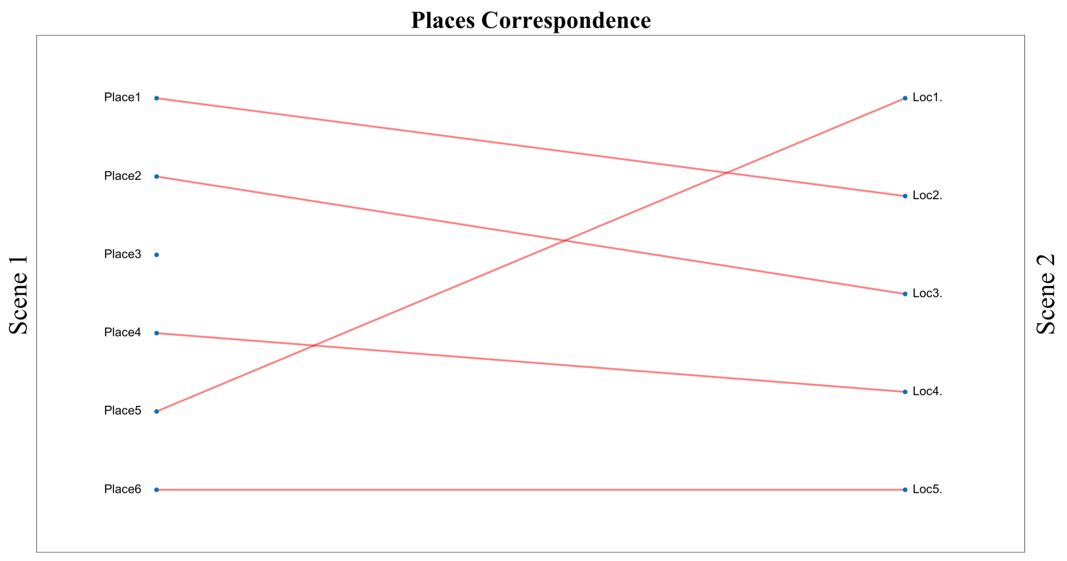


Figure 5.11: Visual representation of *place* matching between navigations, according to closest *place* centroids. For this experiment all clusters found in Navigation2 found a *place* match in Navigation1. However, not all *places* from Navigation1 had a correspondence. This might happen if a different amount of clusters is found or if the closest *place* from Navigation1 is the same for two clusters found in Navigation2, in which case only the closest cluster of Navigation2 is matched to the *place* of Navigation1.

Chapter 6

Conclusion and Future Work

The work described in this thesis aimed to propose a topological mapping strategy using a low-cost mobile robot that could enable the robot to recognize previously mapped areas, and in doing so, locate itself in the topological map. Our approach relies on an estimation of the 3D structure of the environment and the notion that clusters in this structure can be considered sign posts for navigation.

For evaluating our proposal we utilized an in-house built mobile robot equipped with a monocular camera sensor and an low-cost IMU. We have also utilized a public dataset in one of our experiments.

Results seem to suggest that the methodology proposed can provide topological localization for the scenarios reviewed. Even though we only used a monocular camera and an IMU, a sensor suite which has difficulties modeling distances, the 3D structure given by the SLAM system proved enough to create matches between navigations. Recognition of places was thus successful. However, in more challenging scenarios there might be a need to either add another type of sensor that directly observes distances, such as a depth camera.

Some challenges presented themselves during our experiments. Real world locations appeared repeated as different nodes of the map even if they were correctly matched in the second navigation. At times evaluation of the clustering via images revealed that there were outlier locations included in the cluster that should not belong there.

Both of the mentioned issues are influenced by the clustering distance and minimum number of points per cluster parameters. In the future, it might be worthwhile pursuing how to choose these parameters automatically given a certain 3D pointcloud. Another possible avenue for future research is the labelling step of the methodology. Object-detection can be used to give semantic meaning to the found clusters, enabling the robot to have a more human-like understanding of the environment.

The proposed methodology compares the full 3D estimation of each scene to the other which is not efficient. For better scalability and online capabilities, it is worthwhile to pursue a partitioning of the scenes in smaller regions and do local comparisons instead.

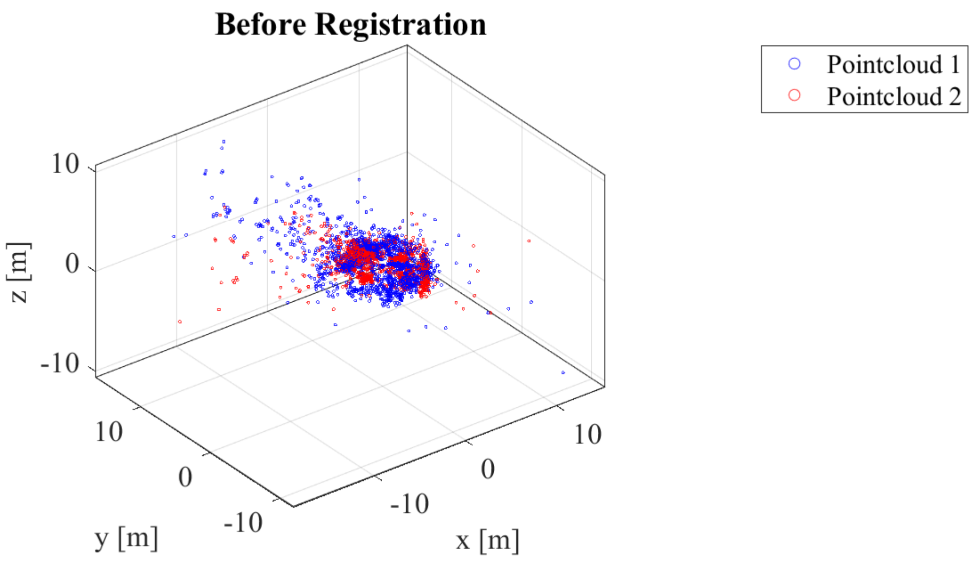
Appendix A

Place Matching Results

In this section we present some results related to each experiment.

A.1 Experiment I

For experiment I we show the pointclouds of Navigation1 and Navigation2 superimposed and centered before and after registration in figure A.1. Registration results were near perfect with Matlab shifting the colours of points between red and blue as we moved the registered pointcloud since they they were on top of each other. We also show the same pointclouds observed from above after clustering in figure A.2. We can see that clusters appeared in the same general location in both pointclouds. However, in Navigation1 the clusters encompassed more points as well as there appearing more clusters. Since the trajectory for Navigation2 was shorter, it could not capture as much information about the environment structure, leading to less datapoints for clustering. We also show the *place* matchings in figure A.3, to showcase how good the match pairings were between navigations. Overall the matches seem correct, with some improvements points that are discussed in section 5.2.



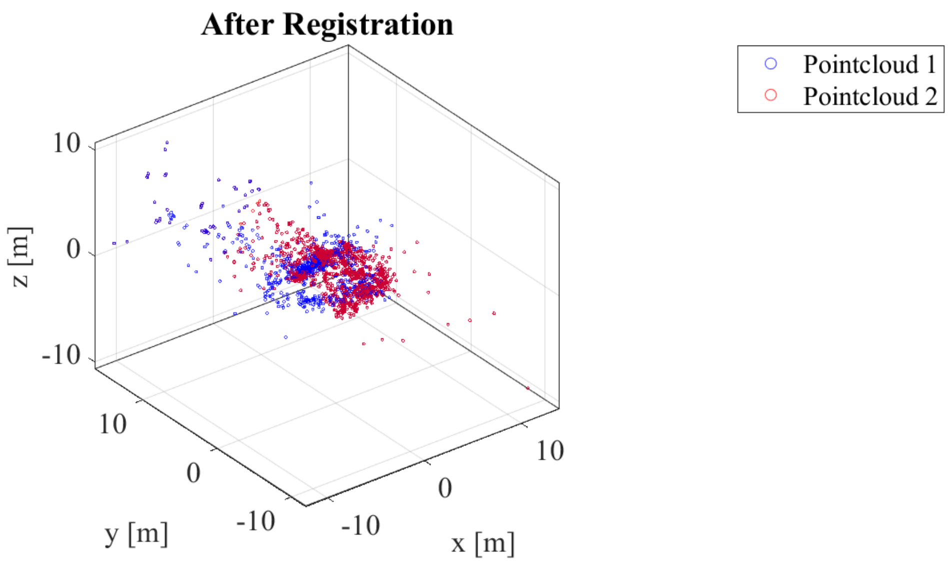
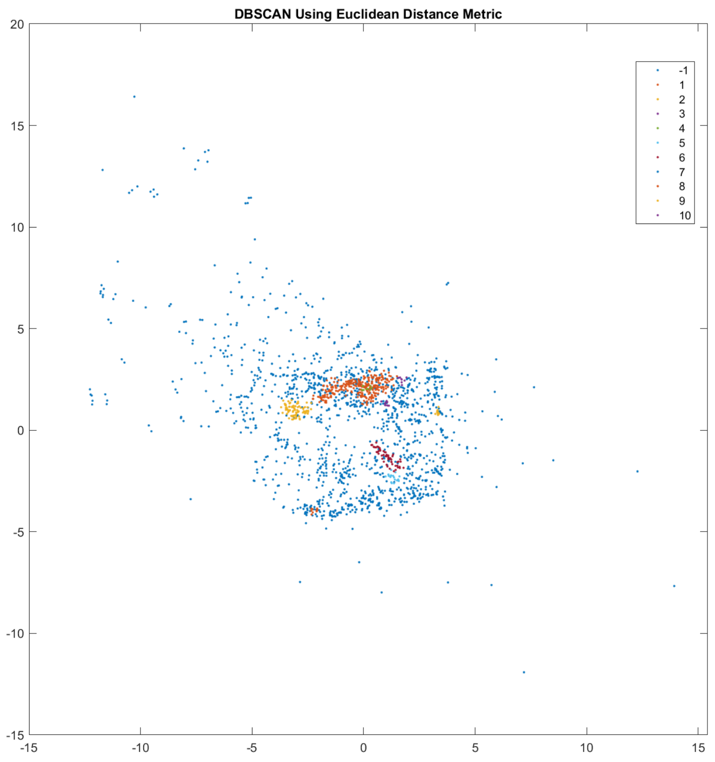


Figure A.1: Pointclouds obtained from Navigation1 (blue) and from Navigation2 (red) for Experiment I. On top we show the superimposed pointclouds without registration. On the bottom we show the pointclouds after registration using plane-to-plane ICP with an inlier threshold of 0.75.



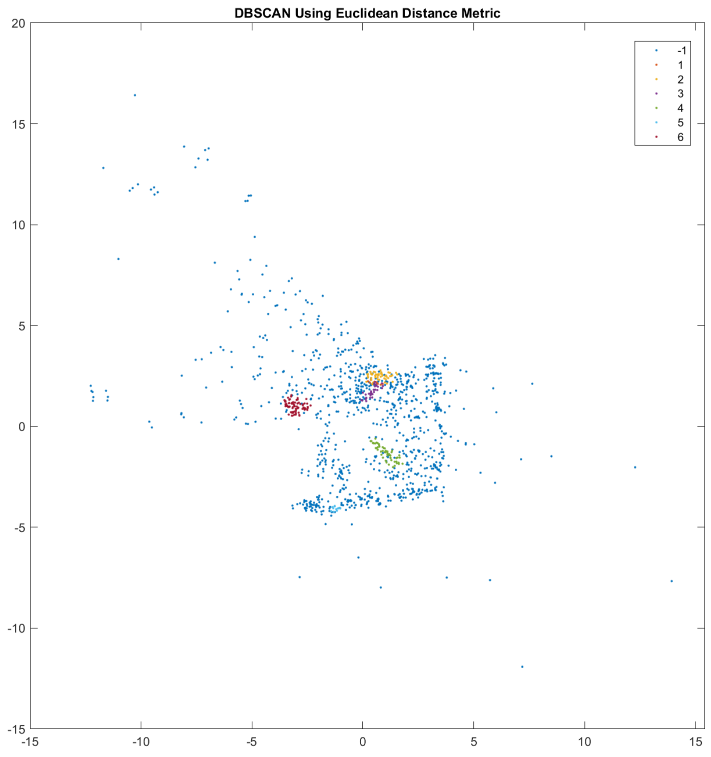


Figure A.2: Above view of pointclouds from Navigation1 (top) and Navigation2 (bottom) after clustering. Colours represent each cluster with dark blue representing outliers. For Navigation1 ten clusters were identified, and six clusters were identified for Navigation2. For both navigations the clustering euclidean distance threshold for DBSCAN was 0.3 and the minimum points per cluster was 11.

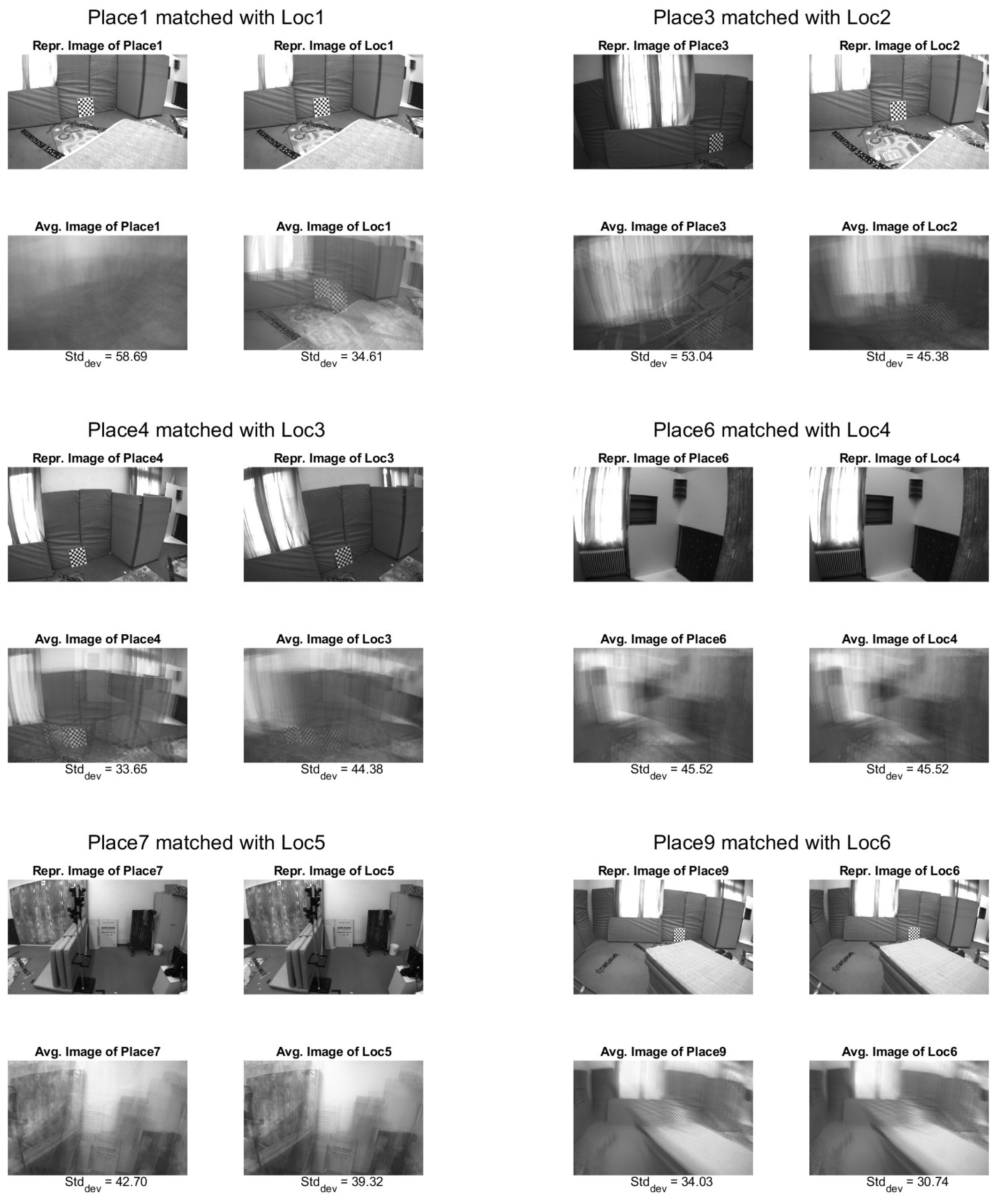
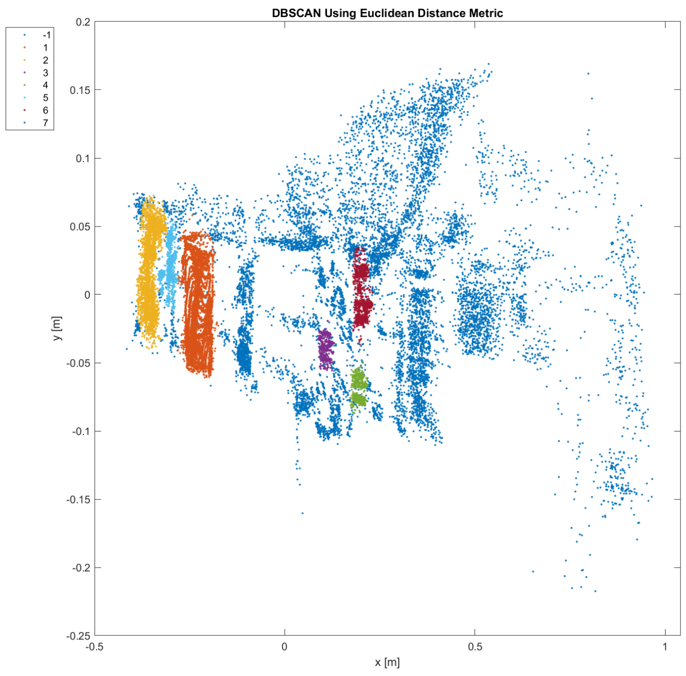


Figure A.3: Reference images for each *place* match for Experiment I. Each match presents two sets of images, images related to the *place* of Navigation1 that was matched (left) and images related to the matched cluster of Navigation2 (right). For each *place* and cluster, two images are shown. A representative image of the cluster, and the average image taken by using all images related to said cluster. The representative image is obtained by finding the closest image in the cluster set to the average image. Below each average image we show the brightness standard deviation found for the cluster.

A.2 Experiment II

For experiment II we show the pointclouds that resulted from each navigation observed from above after clustering in figure A.4. VisualSFM provided more 3D information that the SLAM filter. This lead to more distinct object differentiation in the pointclouds for both navigations. Again in this experiment we verify that clusters seem to appear at the same real world locations as in the previous experiment. We also show the *place* matchings in figure A.5, to showcase how good the match pairings were between navigations. Overall the matches seem correct, with some improvements points that are discussed in section 5.3.



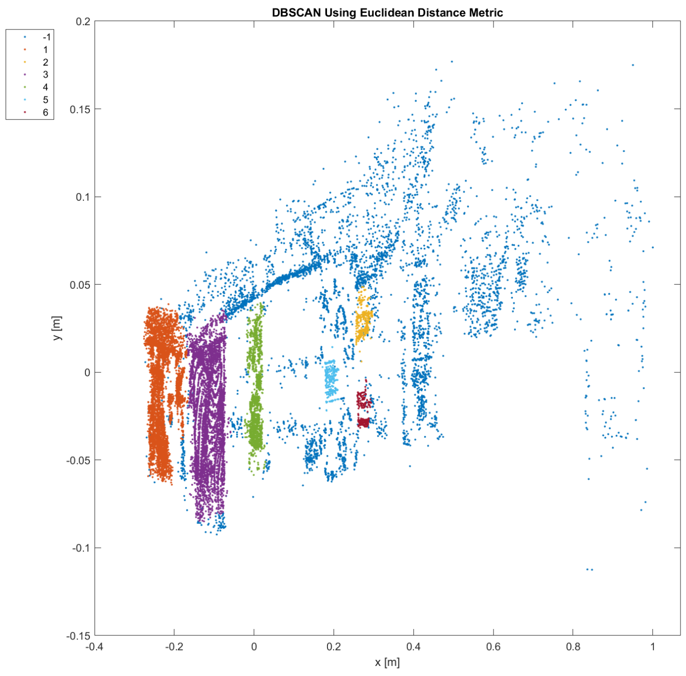


Figure A.4: Front view of pointclouds from Navigation1 (top) and Navigation2 (bottom) after clustering. Colours represent each cluster with dark blue representing outliers. For Navigation1 seven clusters were identified, and six clusters were identified for Navigation2. For both navigations the clustering euclidean distance threshold for DBSCAN was 0.015 and the minimum points per cluster was 200.

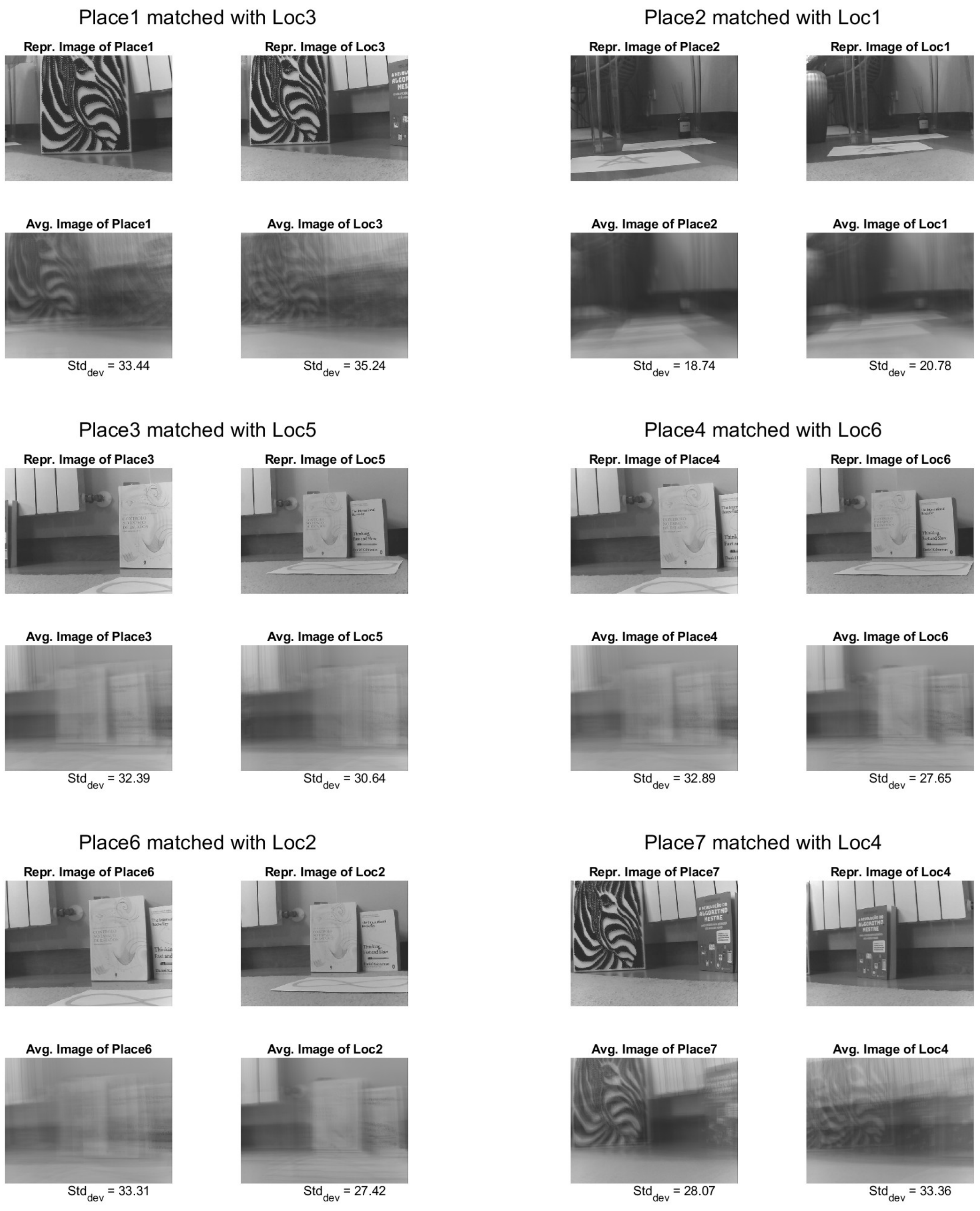
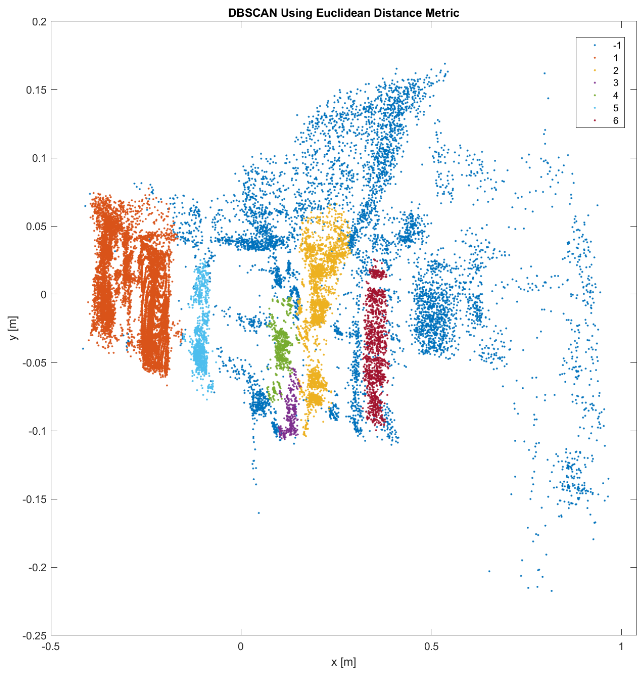


Figure A.5: Reference images for each *place* match for Experiment II. Each match presents two sets of images, images related to the *place* of Navigation1 that was matched (left) and images related to the matched cluster of Navigation2 (right). For each *place* and cluster, two images are shown. A representative image of the cluster, and the average image taken by using all images related to said cluster. The representative image is obtained by finding the closest image in the cluster set to the average image. Below each average image we show the brightness standard deviation found for the cluster.

A.3 Experiment III

For experiment III we show the same plots as for experiment II. First we show the pointclouds that resulted from each navigation observed from above after clustering in figure A.6. Even in the case that navigations took different trajectories observing the same scene it is possible to observe from the same figure that clusters still formed around the same locations. Importantly, for this experiment even though clusters formed around the same location we can observe that it is harder to have clear matchings just from observation of the clustered pointclouds. We also show to place matchings in figure A.7, to showcase how good the match pairings were between navigations. Overall the matches seem correct, with some improvements points that are discussed in section 5.4.



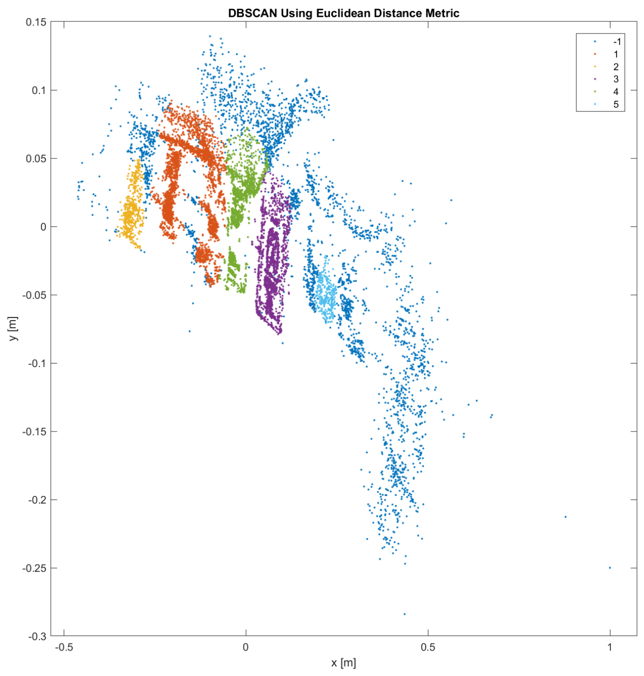


Figure A.6: Front view of pointclouds from Navigation1 (top) and Navigation2 (bottom) after clustering. Colours represent each cluster with dark blue representing outliers. For Navigation1 six clusters were identified, and five clusters were identified for Navigation2. For both navigations the clustering euclidean distance threshold for DBSCAN was 0.025 and the minimum points per cluster was 250.

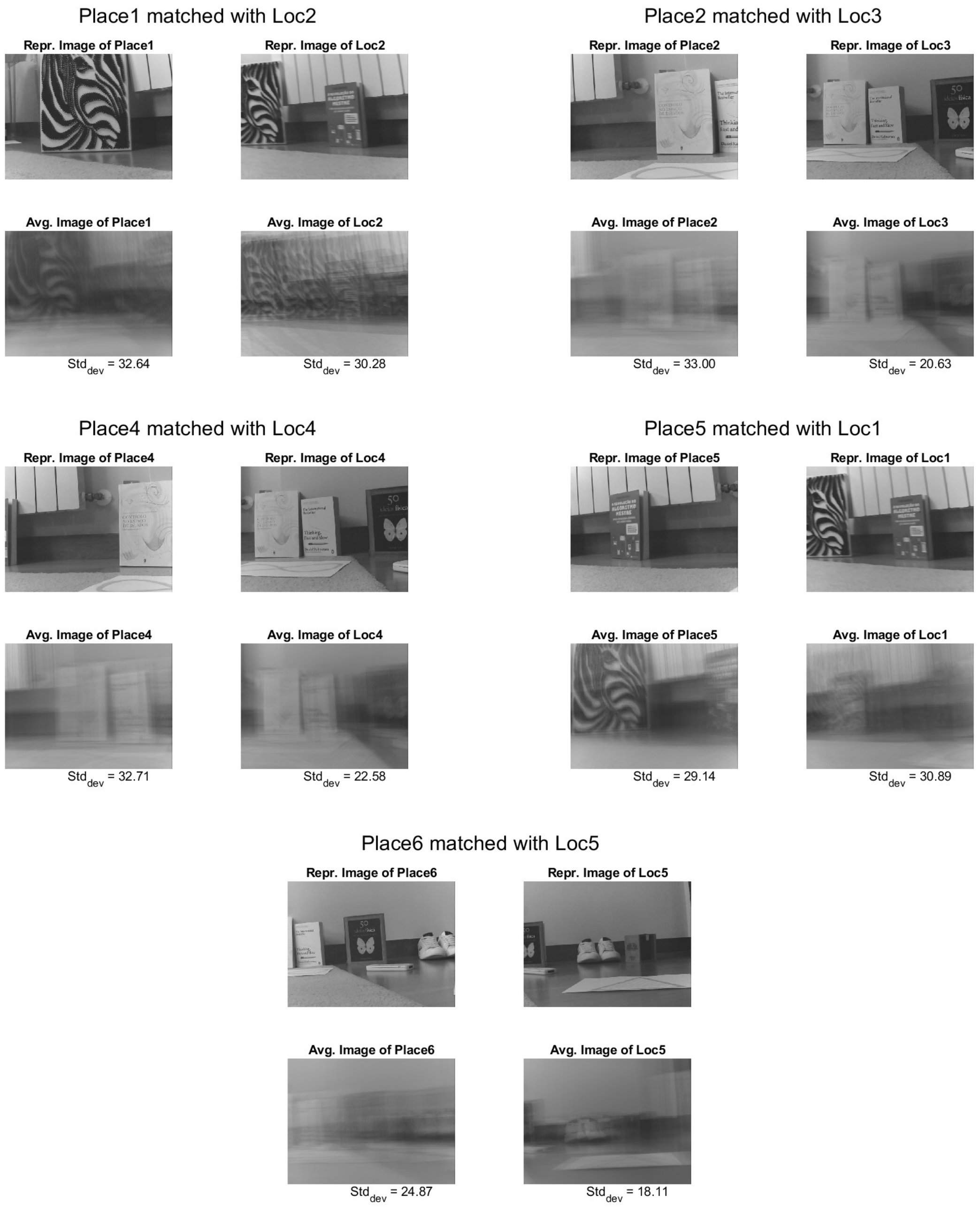


Figure A.7: Reference images for each *place* match for Experiment III. Each match presents two sets of images, images related to the *place* of Navigation1 that was matched (left) and images related to the matched cluster of Navigation2 (right). For each *place* and cluster, two images are shown. A representative image of the cluster, and the average image taken by using all images related to said cluster. The representative image is obtained by finding the closest image in the cluster set to the average image. Below each average image we show the brightness standard deviation found for the cluster.

Appendix B

Nonlinear Systems Filtering Tools, Lie Groups

A Kalman filter can make probabilistic estimations of a system's state based on Gaussian uncertainty and, as such, is widely used in control theory and other disciplines in order to either fuse sensor information or estimate parts of the state that are not available for measurement. If the system is linear the Kalman filter is statistically optimal and converges to the true state.

However, these distributions stop being Gaussian if the system is nonlinear, making the Kalman filter no longer applicable.

The extended Kalman filter (EKF) linearizes the model functions locally, approximating the uncertainty distributions to a local Gaussian. The first order linearization, which the EKF uses, is often very inaccurate to the true probability distribution of the state. This is because most system's have highly non-linear behaviours. This is the case in most mobile robotics applications.

The Unscented Kalman filter features a distinctive methodology based on the unscented transformation. Once again, the uncertainty associated with the state is to be represented as a Gaussian distribution; however, in the Unscented Kalman filter it is depicted by a selected minimal set of weighted χ points. These points are propagated through the system's non-linear functions. The true mean and covariance of the state distribution can be accurately represented by these propagated points.

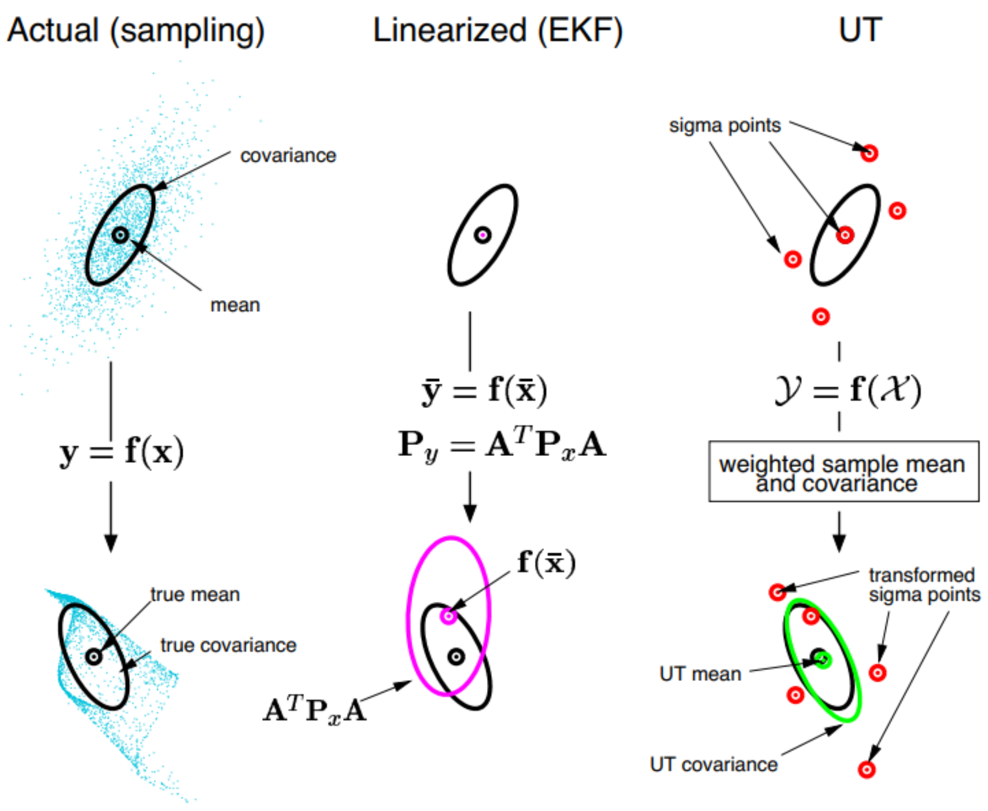

Actual (sampling)
Linearized (EKF)
UT
covariance
mean
sigma points
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
true mean
true covariance
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
transformed sigma points
UT mean
UT covariance

Figure B.1: Comparison between EKF and UKF in terms of uncertainty propagation. Left side shows the real mean and covariance. Middle presents the EKF first order linearization; Right showcases the unscented transformation used by UKF. From [27]

Lie groups are mathematical differentiable manifold groups. The significance of Lie groups like $SO(3)$ and $SE(3)$ in the context of orientation, represented by rotations, and the space of poses, respectively, has long been recognized in the field of robotics [29] as a useful mathematical tool. In the last twenty years, there has been much research on probability distributions on $SE(3)$ and how they might be used for estimation and control [40].

Using Lie groups for state representation and Lie algebra for state propagation has shown numerical consistency advantages compared to standard EKF, it also allows for a more compact representation without the need for numerous representation conversions [2]. We will now introduce the two Lie groups used in the structure of the filter chosen for our work [25].

The Special Orthogonal Group SO(3)

This Lie group comprises all three-dimensional rotations. Every member of this group is a 3×3 matrix. The elements of $SO(3)$ can be defined as follows $SO(3) := \{\mathbf{R} \in \mathbb{R}^{3\times3} : \mathbf{R}\mathbf{R}^\top =$

$\mathbf{I}_3 . det(\mathbf{R}) = 1\}$, where $\mathbf{I}_3$ is the 3×3 identity matrix. Matrix multiplication is the operator within this Lie group. As stated in the above definition, the identity element is the identity matrix $\mathbf{I}_3$, and an inverse element can be derived using the transpose operator.

Each Lie group is connected with a Lie Algebra, which is a unique tangent vector space at the identity element of the Lie group, which is derived through differentiation of the Lie group. Equation B.1 describes a skew-symmetric matrix which can be represented by the operator $[.]_\times$ over a 3D vector $\omega = [\omega_x \; \omega_y \; \omega_z] \in \mathbb{R}^3$.

$$\omega^\wedge = [\omega]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \in \mathfrak{so}(3) \tag{B.1}$$

This operation represents the transformation from $\mathbb{R}^3$ to the Lie Algebra $\mathfrak{so}(3)$ of the Lie group $SO(3)$.

Considering a matrix $[\omega]_\times \in \mathfrak{so}(3)$, we can define the rotation matrix related with this skew-symmetric matrix as $\exp([\omega]_\times) \in SO(3)$. To compute the exponential and obtain the mapping from the Lie algebra $\mathfrak{so}(3)$ to the Lie group $SO(3)$ Rodriges' Formula is used:

$$\mathbf{R} = \exp([\omega]_\times) = \mathbf{I}_3 + \frac{\sin\|\omega\|}{\|\omega\|} + \frac{1 - \cos\|\omega\|}{\|\omega\|^2}[\omega]_\times^2 \tag{B.2}$$

Alternatively, if we wish to map an element from $SO(3)$ to $\mathfrak{so}(3)$, we must apply the logarithmic map. First, we must determine the angle of rotation θ, from the rotation matrix's trace:

$$\theta = \arccos \frac{Tr(\mathbf{R}) - 1}{2} \tag{B.3}$$

using θ, the skew-symmetric matrix can be calculated as:

$$[\omega]_\times = \log \mathbf{R} = \frac{\theta}{2 \sin \theta}(\mathbf{R} - \mathbf{R}^\top) \tag{B.4}$$

The result of converting this matrix back to a 3D vector is a 3D vector of the form ω defined previously. The component values are the original rotation matrix's axis-angle representation. This indicates that this exponential and logarithmic mapping can be utilized to convert between these rotational representations. Because the Lie algebra is the consequence of the differentiation of the Lie group, the transformation from the Lie algebra to the Lie group corresponds to an integration. This means that if we transform a vector of angular velocity $\omega \in \mathbb{R}^3$, into an element of the Lie group $SO(3)$, we obtain the rotation completed over a time frame.

The Special Euclidean Group SE(3)

This group's elements can be defined as $SE(3) := \left\{ \chi = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} : \mathbf{R} \in SO(3), \mathbf{p} \in \mathbb{R}^3 \right\}$, where **0** represents a 1×3 vector of zeros.

These elements, known as homogeneous transformation matrices, represent rigid transformations. The rotation **R** and translation **p** define these transformations. This group's operator is also the matrix multiplication. These structures are utilized as a displacement operator or coordinate system change in various robotics applications. Regarding the Lie Algebra, because it is the consequence of differentiating the Lie group over time, its members can be thought of as angular and linear velocities.

Unscented Kalman Filter Implementation [25]

The filter chosen is an Unscented Kalman filter and it uses a Lie group and its Lie algebra for state representation and state propagation (see in appendix B some Lie groups terminology used in this section). The following sections detail the state structure, the system's model, the sensor's model and state and uncertainty propagation.

State Space

The state space used for this work is an extension of the $SE(3)$ Lie group. This Lie group is defined in $\chi \in SE_{2+p}(3)$ space. The state is composed of all the variables that we want the filter to estimate. It contains the position $x \in \mathbb{R}^3$, $\mathbf{R} \in SO(3)$, linear velocity $\mathbf{v} \in \mathbb{R}^3$ and the 3D positions of landmarks $\mathbf{p}_1, ..., \mathbf{p}_n \in \mathbb{R}^3$ tracked in the scene. The state is formed by a square matrix χ with dimensions $(5+p) \times (5+p)$:

$$\chi = \begin{bmatrix} \mathbf{R} & \mathbf{v} & \mathbf{x} & \mathbf{p}_1 ... \mathbf{p}_n \\ & \mathbf{0}_{(p+2)\times 3} & & \mathbf{I}_{(p+2)\times(p+2)} \end{bmatrix} \tag{B.5}$$

In terms of the operation that defines this group, it constitutes a propagation. This means that if a system experiences a change χ relative to its previous state χ_{t-1}, we can calculate its new state as $\chi_t = \chi * \chi_{t-1}$. Because groups have an inverse operation by definition, we have the option of reversing the state by applying this operation and obtain the previous state.

To have an online estimation of the IMU biases we append these to the state. Consequently, the state has concatenated the IMU biases defined as the bias vector $b \in \mathbb{R}^6$:

$$b = \begin{bmatrix} b_\omega^\top \ b_a^\top \end{bmatrix}^\top \tag{B.6}$$

with the accelerometer bias $b_a \in \mathbb{R}^3$ and gyroscope bias $b_\omega \in \mathbb{R}^3$. Finally, the state of the filter is expressed as the tuple (χ, b).

Dynamic Model

The system model chosen assumes that the body can navigate space in all dimensions and can take any pose. It assumes that the robot is navigating a flat earth and it is equipped with an IMU:

$$\text{body state} = \begin{cases} \dot{\mathbf{R}} = \mathbf{R}(\omega - b_\omega + n_\omega)_\times \\ \dot{\mathbf{v}} = \mathbf{R}(a - b_a + n_a) - g \\ \dot{\mathbf{x}} = \mathbf{v} \end{cases} \tag{B.7}$$

$$\text{IMU biases} = \begin{cases} \dot{b}_\omega = nb_\omega \\ \dot{b}_a = nb_a \end{cases} \tag{B.8}$$

$$\text{landmarks} = \begin{cases} \dot{p}_i = 0, i = 0, ..., p \end{cases} \tag{B.9}$$

where $(\omega)_\times$ portrays the skew-symmetric matrix related with the cross product with vector $\omega \in \mathbb{R}^3$ (as shown in equation B.1). The multiple noises are grouped as:

$$n = [n_\omega{}^\top \ n_a{}^\top \ n_{b\omega}{}^\top \ n_{ba}{}^\top] \sim \mathcal{N}(0,\ Q) \tag{B.10}$$

Sensors Model

The measurement model chosen takes as the calibrated monocular camera frames. The camera observes p landmarks that are tracked in the scene. Each landmark $\mathbf{p}_i$ is observed according to the standard pinhole model and the respective projection model, as such:

$$\mathbf{y}_i = \begin{bmatrix} y^i{}_u \\ y^i{}_v \end{bmatrix} + n^i{}_{\mathbf{y}} \tag{B.11}$$

where $\mathbf{y}_i$ is the result of the projection:

$$\lambda \begin{bmatrix} {y^i}_u \\ {y^i}_v \\ 1 \end{bmatrix} = K[\mathbf{R}^\top{}_{B\to C}(\mathbf{p}_i - \mathbf{x}) - \mathbf{t}_{B\to C}] \tag{B.12}$$

where γ is the scale factor, K is the camera intrinsics matrix, **x** is the IMU/Body position in the world frame, **R** is the IMU/Body orientation, $\mathbf{t}_{B\to C}$ and $\mathbf{R}^\top{}_{B\to C}$ are respectively translation and rotation from the IMU to the camera in the world frame.

With this projection a landmark $\mathbf{p}_i$ is converted from the world frame to the image plane and is now compared to its corresponding 2D feature that is being tracked frame to frame. In case a certain 3D landmark projection and its respective 2D feature are too distant that 3D landmark is invalidated.

Time Discretization

Since the filter is run computationally it has to be discretized in time. This implementation uses the standard Euler method to discretize the dynamic system's equations, for a small step Δt as such:

$$\text{body state} = \begin{cases} \mathbf{R}_{t+\Delta t} = \mathbf{R}_t \exp[(\omega_t - b_{\omega,t})\Delta t + \mathbf{Cov}(n_\omega)^{1/2} g\sqrt{\Delta t}]_\times \\ \mathbf{v}_{t+\Delta t} = \mathbf{v}_t + (\mathbf{R}_t(a_t - b_{a,t}) - g)\Delta t \\ \mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{v}\Delta t \end{cases} \tag{B.13}$$

Uncertainty of the state on Lie Groups

Given the state representation defined in equation B.5, the uncertainty of the state is defined as a probability distribution $\chi \sim \mathcal{N}(\overline{\chi},\, \mathbf{P})$:

$$\chi = \mathbf{Exp}(\xi)\overline{\chi}, \mathcal{N}(0,\, \mathbf{P}) \tag{B.14}$$

with

$$\mathbf{Exp}(\xi)\overline{\chi} = \mathbf{exp}([\xi]_\times) \tag{B.15}$$

where **Exp** maps from the vector space to the Lie group, **exp** maps from the Lie algebra to the Lie group and $[.]_\times$ from the vector space to the Lie algebra. The uncertainty ξ is defined as $\xi = [\xi_\mathbf{R}{}^\top\ \xi_\mathbf{v}{}^\top\ \xi_\mathbf{x}{}^\top\ \xi_{\mathbf{p}_i}{}^\top\ ...\ \xi_{\mathbf{p}_n}{}^\top]^\top$. To transform the uncertainty ξ from the vector space to the Lie Algebra we use the transformation:

$$[\xi]_\times = \xi^\wedge = \begin{bmatrix} [\xi_{\mathbf{R}}]_\times & \xi_{\mathbf{v}} & \xi_{\mathbf{x}} & {\xi_{\mathbf{p}_i}}^\top \dots & {\xi_{\mathbf{p}_n}}^\top \\ & \mathbf{0}_{(2+p)\times(5+p)} & & & \end{bmatrix} \tag{B.16}$$

Appendix C

MonoSLAM parametrization and Feature Initialization

In this chapter we detail further background on MonoSLAM [11] [8]. We detail a different depth parametrization proposed in [8], that constituted an update for MonoSLAM [11]. We also detail how both the original MonoSLAM implementation and the new depth parametrization improved version initialized features.

C.1 Inverse Depth Parametrization

Inverse depth parametrization - an improvement for MonoSLAM [8]

In previous research works, including the original MonoSLAM [11], there were problems integrating features with large depth. This is due to the small parallax compared to the translation of the system which needs to be large enough for an accurate depth estimate.

In 2008 a new parametrization technique, inverse depth parametrization, is proposed in [8] which solves this problem, integrating a feature whichever its depth and gathering new information as soon as it is detected.

The standard representation for scene points i in terms of Euclidean XYZ coordinates is:

$$\mathbf{x}_i = \begin{bmatrix} X_i & Y_i & Z_i \end{bmatrix}^\top \tag{C.1}$$

The proposed update to the parametrization is:

$$\mathbf{y}_i = \begin{bmatrix} x_i & y_i & z_i & \theta_i & \phi_i & \rho_i \end{bmatrix}^\top \tag{C.2}$$

Which models a 3-D point located at:

$$\mathbf{x}_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \phi_i) \tag{C.3}$$

$$\mathbf{m} = (cos\phi_i sin\theta_i,\ -sin\phi_i,\ cos\phi_i cos\theta_i)^\top \tag{C.4}$$

The y_i vector encodes the ray from the first camera position from which the feature was observed by x_i, y_i, z_i, the camera optical center, and θ_i, ϕ_i azimuth and elevation (coded in the world frame) defining unit directional vector $\mathbf{m}(\theta_i, \phi_i)$. The point's depth along the ray d_i is encoded by its inverse $\rho_i = 1/d_i$.

This new parametrization requires a new measurement model:

$$\mathbf{h}^C = \mathbf{h}^C_\rho = \mathbf{R}^{CW} \left[\rho_i \left[\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} - \mathbf{r}^{WC} \right] + \mathbf{m}(\theta_i, \phi_i) \right] \tag{C.5}$$

where the directional vector has been normalized using the inverse depth. It is worth noting that (11) can be safely used even for points at infinity, i.e., $\rho_i = 0$ an advantage compared to the previous parametrization, although not the only one.

C.2 Feature Initialization

Original MonoSLAM

Initialization of new features in the original 2007 paper [11] is done by initializing a semi-infinite line starting at the camera position where the feature was first observed and extending it to infinity along the feature viewing direction and using a Gaussian uncertainty for the depth parameter. In the first observation the uncertainty is high, but with each subsequent observation more information is obtained and the uncertainty on the feature's depth decreases. When the depth uncertainty is below an acceptable level the feature is considered fully initialized and the information it provides is finally available for estimating the state. In this model, uncertainty on the feature's initial viewing direction is disregarded since this uncertainty is low compared to the depth uncertainty.

Updated parametrization - Inverse depth

Initialization of new features in the updated algorithm of the 2008 article is similar to the previous' year proposal. A feature is also represented by the camera pose with which it was first observed. The difference is that its depth is now parametrized as inverse depth, (8) and (9), which is proved in [8] to have better linearity properties, even at high uncertainty, than the standard representation (7). This allows for the use of the information provided by each feature as soon as it is initialized since it will not perturb the linearity assumption of the filter. At first it can only provide information on the orientation of the camera and not its translation but with more observations of the feature with enough parallax translation information is also conveyed.

Bibliography

[1] I. Armeni, Z.Y. He, A. Zamir, J. Gwak, J. Malik, M. Fischer, and S. Savarese. 3D scene graph: A structure for unified semantics, 3D space, and camera. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5663–5672, 2019.

[2] A. Barrau and S. Bonnabel. An EKF-SLAM algorithm with consistency properties, 2015.

[3] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart. Topomap: Topological mapping and navigation based on visual SLAM maps. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3818–3825, 2018.

[4] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35, 01 2016.

[5] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[6] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta. Neural topological slam for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[7] J. Civera, D. Gálvez-López, L. Riazuelo, J. D. Tardós, and J. M. M. Montiel. Towards semantic SLAM using a monocular camera. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1277–1284, 2011.

[8] Javier Civera, Andrew J. Davison, and J. M. MartÍnez Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945, 2008.

[9] J. Cruz. Wireless mobile camera, Master's Dissertation. *Institute for Systems and Robotics, Instituto Superior Técnico / UL, Lisbon, Portugal*, 2015.

[10] A.J. Davison. Active search for real-time vision. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 66–73 Vol. 1, 2005.

[11] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[12] K. Eckenhoff, P. Geneva, and G. Huang. Closed-form preintegration methods for graph-based visual–inertial navigation. *The International Journal of Robotics Research*, 38(5):563–586, Apr 2019.

[13] R. Elvira, J. D. Tardós, and J. M. M. Montiel. ORBSLAM-atlas: a robust and accurate multi-map system, 2019.

[14] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[15] M. Ferrera, A. Eudes, J. Moras, M. Sanfourche, and G. Le Besnerais. OV^{2}SLAM: A fully online and versatile visual SLAM for real-time applications. *IEEE Robotics and Automation Letters*, 6(2):1399–1406, 2021.

[16] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration for real-time visual-inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017.

[17] P. Furgale, J. Rehder, and R. Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 2013.

[18] Emilio Garcia-Fidalgo and Alberto Ortiz. Vision-based topological mapping and localization methods: A survey. *Robotics and Autonomous Systems*, 64:1–20, 2015.

[19] P. Geneva, K. Eckenhoff, and G. Huang. A linear-complexity ekf for visual-inertial navigation with loop closures. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3535–3541, 2019.

[20] E. Giacoumidis, Y. Lin, M. Jarajreh, S. O'Duill, K. McGuinness, P. F. Whelan, and L. P. Barry. A blind nonlinearity compensator using DBSCAN clustering for coherent optical transmission systems. *Applied Sciences*, 9(20), 2019.

[21] Guoquan Huang. Visual-inertial navigation: A concise review. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9572–9582, 2019.

[22] H. Jo, H. M. Cho, S. Jo, and E. Kim. Efficient grid-based Rao–Blackwellized particle filter SLAM with interparticle map sharing. *IEEE/ASME Transactions on Mechatronics*, 23(2):714–724, 2018.

[23] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34, 02 2014.

[24] T. Lupton and S. Sukkarieh. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Transactions on Robotics*, 28(1):61–76, 2012.

[25] S. Bonnabel M. Brossard and A. Barrau. Unscented Kalman filtering on Lie groups for fusion of IMU and monocular vision. *International Conference on Robotics and Automation (ICRA)*, 2018.

[26] Z. Murez, T. As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich. Atlas: End-to-end 3D scene reconstruction from posed images. In *ECCV*, 2020.

[27] P. Nogueira. Visual inertial odometry for mobile home robots, Master's Dissertation. In *Instituto Superior Técnico / UTL, Lisbon, Portugal*, 2021.

[28] L. Oth, P. Furgale, L. Kneip, and R. Siegwart. Rolling shutter camera calibration. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1360–1367, 2013.

[29] F. C. Park, J. E. Bobrow, and S. R. Ploen. A Lie group formulation of robot dynamics. *The International journal of robotics research*, 14(6):609–618, 1995.

[30] T. Qin, P. Li, and S. Shen. VINS-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[31] T. Qin, J. Pan, S. Cao, and S. Shen. A general optimization-based framework for local odometry estimation with multiple sensors, 2019.

[32] J. Redmon and Rand Farhadi A. Divvala, Sand Girshick. You Only Look Once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[33] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[34] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. Maplab: An open framework for research in visual-inertial mapping and localization. *IEEE Robotics and Automation Letters*, 3(3):1418–1425, 2018.

[35] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[36] G. J. Stein, C. Bradley, V. Preston, and N. Roy. Enabling topological planning with monocular vision. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1667–1673, 2020.

[37] A. Vale. *Mobile Robot Navigation in Outdoor Environments: A Topological Approach*. PhD thesis, Universidade Técnica de Lisboa, 2005.

[38] K. J. Wu, C. X. Guo, G. Georgiou, and S. I. Roumeliotis. VINS on wheels. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5155–5162, 2017.

[39] Z. Yu, L. Zhu, and G. Lu. VINS-motion: Tightly-coupled fusion of VINS and motion constraint. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7672–7678, 2021.

[40] M. Zefran, V. Kumar, and C. Croke. Metrics and connections for rigid-body kinematics. *International Journal of Robotics Research*, 18(2):243–258, 1999.