UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JOSIAS ALEXANDRE OLIVEIRA

Long-Term Map Maintenance in Complex Environments

Vitória,ES

Jun 6, 2021

Josias Alexandre Oliveira

Long-Term Map Maintenance in Complex Environments

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para obtenção do título de Mestre em Informática.

Supervisor: Claudine Badue

Co-supervisor: Alberto Ferreira De Souza

Vitória,ES

Jun 6, 2021

Acknowledgements

Agradeço a minha orientadora Profa. Doutora Claudine Badue, ao meu co-orientador Prof. Alberto Ferreira De Souza, à CAPES e a todos do LCAD.

Resumo

Os veículos autônomos devem observar as mudanças no ambiente externo e refletir tais mudanças em representações internas (por exemplo, mapas) afim de garantir um comportamento adequado e segurança. Como as mudanças nos ambientes externos são inevitáveis, é desejável um sistema de mapeamento para os robôs móveis que dependam de mapas e que objetivam operação autônoma de longo prazo. Neste trabalho, propomos um novo sistema de mapeamento em larga escala para o Intelligent Autonomous Robotic Automobile (IARA). O novo sistema de mapeamento é baseado no algoritmo GraphSLAM, com extensões para lidar com a calibração da odometria diretamente na otimização do grafo e a mesclagem e manutenção de mapas a longo prazo. O sistema de mapeamento pode usar dados de sensor de um ou vários robôs para construir e mesclar mapas de grade de ocupação. O desempenho do sistema é avaliado em uma série de experimentos realizados com dados capturados em cenários complexos do mundo real. Os resultados indicam que o novo sistema de mapeamento pode fornecer mapas de grade de ocupação para navegação e localização de veículos autônomos.

Palavras-chave: Mapeamento, SLAM, veículos autônomos

Abstract

Autonomous vehicles should capture the external environment changes into internal representations (for example, maps) for proper behavior and safety. As changes in external environments are inevitable, a lifelong mapping system is desirable for autonomous robots that rely on maps and aim at long-term operation. In this work, we propose a new large-scale mapping system for the Intelligent Autonomous Robotic Automobile (IARA) or any other autonomous vehicle. The new mapping system is based on the GraphSLAM algorithm, with extensions to deal with the calibration of odometry directly in the optimization of the graph and to address map merging for long-term map maintenance. The mapping system can use sensor data from one or more robots to build and merge different types of occupancy grid maps. The system's performance was evaluated in a series of experiments carried out with data captured in complex real-world scenarios. The experimental results indicate that the new large-scale mapping system can provide high-quality occupancy grid maps for later navigation and localization of autonomous vehicles.

Keywords: Mapping, SLAM, Autonomous Vehicles

List of Figures

List of Tables

Contents

1 Introduction

Localization-based Autonomous Vehicles (AVs) typically use maps of the environment to localize themselves and to safely navigate in it. In this context, maps are internal models that robots can use to represent the external world. Robots can use maps to recover spatial models of the environment from their sensors. AVs can localize themselves in relation to the map using their sensor measurements (localization problem). When there is no map, they can create a new one also using their sensors. Vehicle poses can be known when creating the map (mapping problem) or can be estimated as the map is built (simultaneous localization and mapping - SLAM - problem [1]).

One of the requirements imposed on AVs is the ability to operate for a long period of time without human intervention. Therefore, AVs must display a life-long mapping feature that updates their internal maps whenever the environment changes. The changes in the environment can be related to slightly dynamic objects (for example, parked vehicles and traffic cones) and highly dynamic objects (for example, pedestrians and other vehicles). In addition, the environment may undergo even slower modifications to the infrastructure (for example, changes on the road and new buildings). Hence, AVs should also be able to provide high quality maps with clear distinction between static and moving objects, highly or slightly dynamic, or even maps containing only static objects, as in the case of maps for later navigation and localization.

Maps can also change due to different viewpoints and occlusions. For example, a change in the pose of a sensor can modify the way the AV perceives the environment. These issues, altogether, pose the life-long mapping problem, which usually requires merging the current map already built with the new ones from another surveying mission with a single robot or multiple robots at the same time. Since merging maps is a difficult task, it is often called a map merging problem, which consists of combining two or more individual maps without a common frame of reference into a larger global map.

One of the main challenges in the map merging problem is illustrated in Fig. 1, where red and green pixels represent obstacles reached by a LiDAR sensor and projected in the ground. Red pixels represent obstacles in a map while green pixels represent obstacles in another. Considering the final map after improper merging, there is prominent misalignment between the two individual maps.

We have developed an AV, named Intelligent Autonomous Robotic Automobile (IARA, Fig. 2), whose autonomy system follows the typical architecture of self-driving cars [2]. IARA is based on a Ford Escape Hybrid adapted with a variety of sensors and processing units. Its autonomy system is composed of many subsystems, which includes

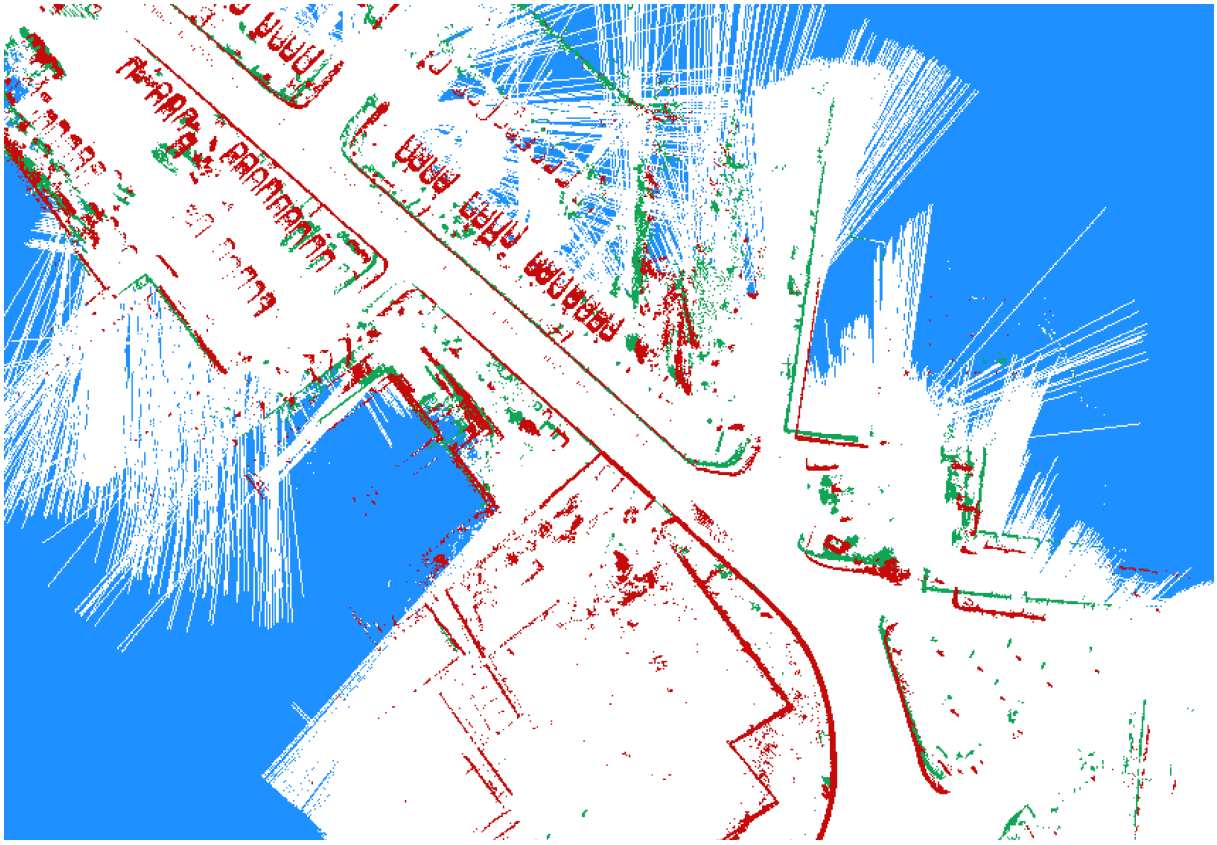


Figure 1 – Map merging.
Example of erroneous map merging. The lack of loop closure between the sessions results in misalignment between the two individual maps.

a Mapper [3], a Localizer [4], a Moving Obstacle Tracker [5], a Traffic Signalization Detector [6,7], a Route Planner, a Path Planner, a Behavior Selector, a Motion Planner [8], an Obstacle Avoider [9] and a Controller [10], among others.

The previous mapping system of IARA was not capable of merging maps from multiple sessions without some manual work. In some cases, the proper merging was not even possible, specially in the presence of GPS signal degradation. In this work, we propose a new mapping system based on the GraphSLAM algorithm for solving the map merging problem in extremely large outdoor environments for IARA or any other autonomous vehicle. The proposed mapping system aims to provide high quality occupancy grid maps for autonomous driving. It is also able to calibrate the vehicle odometry using a new method applied in the graph optimization. The system receives as input one or more log files containing the data from various sensors collected by surveying missions (or sessions) from a single or multiple vehicles, and it outputs the AVs' poses in a global coordinate frame, the parameters of the odometry calibration for each vehicle and different types of grid maps.

We evaluated the performance of the proposed mapping system on logs containing high and slight dynamics in real-world scenarios. The logs were collected using IARA over

Figure 2 – IARA
The Intelligent Autonomous Robotic Automobile (IARA).

a period of two years. The new calibration method was compared against another approach based on Particle Swarm Optimization (PSO) that is available in IARA's autonomy system. The results indicate that the new method provides better calibration values in all logs. The new mapping features were also analyzed through mapping and map merging experiments. The results show that the proposed system is able to correctly estimate the AV poses in a global reference frame, and to correctly merge local maps without drifts and misalignment for all logs.

1.1 Motivation

This work is connected to a research project of the Laboratório de Computação de Alto Desempenho (High Performance Computing Laboratory - LCAD) of the Departamento de Informática (Department of Informatics - DI) at the Universidade Federal do Espírito Santo (Federal University of Espírito Santo - UFES). The project started due to need of improvements in the previous mapping system used by IARA. The previous system presented some limitations that should be addressed in this work. The previous system was not able to handle some scenarios and it lacked the map merging capabilities that we present in this project.

Alongside the internal motivation, the authors also considered to publicize a long-term map maintenance pipeline for AVs. In our case, the main output of the system is an

occupancy grid map with significant resolution for later navigation and localization. Most similar works are related to internal environments, drones and other kind of robots.

1.2 Objectives

The main objective of this project is to improve the previous IARA's mapping system. The new system should be able to perform long-term map maintenance and to handle more complex and even larger scenarios. This project should also contribute to other research projects that depends on the resulting mapping system.

1.3 Contributions

The main contribution of this work is a new mapping system that is capable of building and merging occupancy grid maps with high quality for later navigation and localization. When compared to the previous mapping system, one can observe the following improvements: no need to synchronize the sensors, map merging capabilities, additional LiDAR odometry edges are present in the hypergraph, a new method for odometry calibration that showed better results and a pre-processing step to remove errors and undesired measurements

Other minor contribution is a custom LiDAR odometry (LO) method that executes a simplified semantic segmentation. The new LO method accumulates the previous point clouds to enhance the point cloud registration and it employs a 3D grid filter to reduce the size of the accumulated cloud to avoid memory issues.

1.4 Organization

The remaining are organized as follows. In Chapter 2, we present related work. In Chapter 3, we present the theory behind the simultaneous localization and mapping problem, the GraphSLAM algorithm and the occupancy grid maps. In Chapter 4, we detail the proposed mapping system. In Chapter 5, we describe the experimental methodology and, in Section 6, we discuss experimental results. Finally, in Chapter 7, we close with our conclusions and directions for future work.

2 Related Work

Progress on the SLAM problem can be measured in terms of a wide variety of algorithms, techniques and theoretical frameworks that have been developed in last decades and still remain unresolved questions, such as robustness and outlier rejection for long-term operations [1].

The map merging problem addresses theses open issues in different context application, either with multiple robots running synchronously in collaborative single surveying mission or with only one robot gathering sensory data during multiple surveying missions. While the first is focused in collaborative strategies to accomplish efficient mapping missions, the latter is oriented to long-term mapping and merging strategies in order to accommodate infrastructural changes along time. Both are facets of the same problem but the long-life merging approach can be seen as a more general approach and could also be applied for multi-robot mapping under some assumptions. Both share the same main goal which is to produce a final aggregated map alongside the robot trajectories for all running robots and surveying sessions.

Although some may consider single-robot SLAM in 2D environments as a largely solved problem, research on multi-robot SLAM methods in 3D outdoor mapping is still ongoing, as surveyed by Saeedi et al. [11] four years ago and also more recently, as shown in Section 2.1. SLAM systems are of paramount importance for self-driving cars. As surveyed by Bresson et al. [12], autonomous driving requires that SLAM solutions handle many miles of sensory data without drifting so as the car can navigate in dynamic environments and different weather conditions. The authors also listed some criteria that should be attended by any SLAM system in the context of autonomous driving. Among these, the system's availability criterion does not apply to our work since our system, as a full SLAM method, is run offline in order to create accurate maps for navigation and localization purposes. All other criteria are covered in some extent in our work. In Section 2.2, a more detailed comparisson of our method with the state-of-the-art is presented for single-robot multi-session.

2.1 Multi-Robot Single-Session

The common approach in multi-robot scenarios is to fetch all maps using map-matching to find the relative map transformations and make the merge. Given the computed transformations, display the final map and the trajectory of each robot in that map. Some assumptions are required for this to work, such as knowing the initial positions of robots and a static environment.

A system for real-time multi-robot collaborative SLAM is presented in Deutsch et al. [13]. The distributed robots use their own SLAM system to build local maps, but they also share information (features from images) with a central system, which builds a global pose graph including information from all robots. After the graph optimization is finished, the central system sends the results to all robots, so they keep a consistent local position and map estimates. In contrast, our mapping solution is designed for a posteriori autonomous navigation within maps that are larger than usually found for indoor robots. Ours can handle any map size as it depends only in the available storage.

Differently, Sun et al. [14] convert the map merging into an image registration problem. When two already built grid maps contains overlapping areas, their approach tries to find the best transformation which aligns the maps. In the first step, the algorithm extract features from both maps using the Harris corner detector and it uses the selected features to calculate a initial transformation matrix. In the second step, it applies an iterative algorithm to refine the transformation matrix in order to improve the map merging. In contrast, our method finds the common reference frame through the graph optimization by making usage of GPS and LiDARs sensors, hence the proper alignments are achieved before the map construction.

A system for collaborative Visual-SLAM is presented in [15]. In their system, each robot uses the Extended Kalman Filter (EKF) algorithm for real-time SLAM using cameras and IMU sensors. In parallel, each robot also runs a graph SLAM for global localization improvements. The robots build and share their own local 3D submaps with any team mate in the vicinity. The submaps are merged through an Iterative Closest Point (ICP) using proper 3D feature descriptors. Their system is able to run on low-end processing boards and is meant for robots operating in smaller-size environments when compared to our work. Another difference is the employment of the map matching in the already built 3D map. Our method finds the correct global reference frame before the 2D maps are built as described before.

Similar to [15], the method described in Lazaro et al. [16] relies on sharing data between the robots in the same environment. The authors propose an efficient communication method between the robots in order to share minimal data. The robots share a condensed version of their own graphs to allow for the merging of maps coming from different robots. In the current iteration, our method is meant for a posteriori navigation in high definition maps so we can make usage of all available information in multiple sessions.

Kim et al. [17] provided an extension to the Incremental Smoothing And Mapping (iSAM) algorithm in order to allow multiple-robot mapping. As in our current work, each robot or session generates a separated pose graph, but their proposal maintains the pose graphs in their own local coordinate frame. The relative pose graphs are optimized together

using a special anchor node which connects pairs of pose graphs. These special anchor nodes contains the actual transformations of each pose in the related pose graph into a global coordinate frame. The anchor nodes are created when the robots face each other or when two robots perceive the same feature in the environment. The authors evaluate their system with small wheeled robots and drones in an indoor environment.

The work in Mangelson et al. [18] uses the same concept of anchor nodes as in [17]. Their system shows the same multi-robot SLAM capabilities and the authors provide a new algorithm for robust selection of loop closures between pairs of robots. The authors defined a metric to determine the largest subset of measurements that are pairwise consistent. The consistency between pairs of measurements rejects potential outliers and outperforms equivalent selection methods like Dynamic Covariance Scaling (DCS) [19], Single-Cluster Spectral Graph Partitioning (SCGP) [20] and RANSAC. After finding the largest set, the selected inter-map loop closures are inserted in the graph and the full map is generated.

2.2 Single-Robot Multi-Session

A SLAM system using Normal Distribution Transform (NDT) maps is presented in Einhorn and Gross [21]. The NDT maps are independent from the sensor types and also allows the system to deal with different dimensions in the generated submaps. The authors propose a modification in the NDT mapping to handle free-space measurements which makes them better suited for SLAM. The map merging is achieved by the alignment of the NDT maps through proper registration techniques which are closely related to ICP. The registration algorithm defines a minimization problem which can be solved by Levenberg-Marquardt (LM) and PSO methods in different levels of the NDT maps.

Bonanni et al. [22] present a SLAM method that considers the map as a graph of point clouds. In order to merge two maps, the method visits each node in the first graph and tries to find the best candidate in the second graph. The Normal Iterative Closest Point (NICP) [23] is used to find the registration between the point clouds. The method is similar to our solution as the point clouds are used to provide the correct relationship between the sessions. We use the GPS sensor and a nearest neighbor algorithm to find the best candidates in the second graph.

In Lazaro et al. [24], the authors propose a SLAM system focused on the management of long-term situations. The system is capable of handling both highly dynamic environments and map updates between the sessions. The local maps are represented by nodes in an sparse graph and the spatial relations between adjacent local maps are embedded in the edges of the graph. They use the NICP algorithm to align newer point clouds into the local maps. As the process evolves and more resources are required, some submaps are combined in a new node and the outdated nodes containing the previous

maps are removed. Although the system was evaluated in datasets containing long periods of operation (10 hours), these datasets contain only small indoor environments.

Tanaka [25] proposes a map matching approach with main novelties. Firstly, a retrieval stage uses a visual place recognition (VPR) algorithm to find the loop closures between the sessions. The VPR uses discriminative deep convolutional network (DNCC) features from a specific layer of the neural network. The retrieval stage also estimates a good initial set of map alignment hypotheses. After the initial hypotheses, a second re-ranking stage jointly deforms the maps in order to minimize the differences in shape between them.

Jiang et al. [26] propose a fast map matching method using scan-to-map techniques to improve the odometry. The robot executes a local 2D SLAM using laser scans. Each input scan is matched against a local map using Correlative Scan Matching (CSM) and Point-to-Line Iterative Closest Point (PL-ICP). To merge the submaps their system treats the submaps as images and then uses Shi-Tommasi corner points to build the so-called *triangle features*. After collecting a set of *triangle features*, the system poses the problem of finding the rigid body transform between the submaps as a least-squares estimation.

The work of Ding et al. [27] is meant for large scale outdoor environments, as ours, and use 3D LiDAR for loop closure whilst we use GPS. Their system is capable of building maps using multi-session data as ours. Their system splits the map into submaps and makes usage of an efficient method for loop closure using feature vectors and a loop closure database using the kd-tree algorithm. Their method also takes care of loop closure outliers and actively detects changes in the environment (high and low dynamics).

3 Theoretical Background

This chapter presents the main concepts and algorithms related to this project.

3.1 The Simultaneous Localization and Mapping Problem

The simultaneous localization and mapping problem consists of building or merging a map m of an unknown environment while performing the robot's localization in order to estimate the poses $x_{1:t}$ that describe the robot's path. To accomplish this task, the robot has only the measurements $z_{1:t}$ and the commands $u_{1:t}$. Using a probabilistic approach, one can solve the SLAM on two main forms. The problem can be considered as the posterior over the current pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3.1}$$

where x_t is the pose at time t.

The first form is known as the *online* SLAM problem. The goal of the *online* SLAM is to obtain the current robot location, therefore, the past locations are ignored or discarded. Since this approach discards the previous poses, it does not take advantage of loop closures to estimate the current pose. A loop closure occurs when the robot visits the same place multiple times.

The second form is known and the *full* SLAM, meaning that the entire path should be estimated in the posterior:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.2}$$

where $x_{1:t}$ is the entire path.

This second method is able to revisit previous poses and handle loop closures but calculating the posterior 3.2 is usually hard, so the practical SLAM algorithms rely on approximations [28]. The GraphSLAM algorithm is one of the possible solutions for the SLAM problem and it is the base of our project.

3.2 The GraphSLAM Algorithm

The GraphSLAM algorithm solves the full SLAM problem in an offline method. The SLAM problem is reduced to a maximum likelihood estimation where the robot poses and the map are the main variables to be estimated. The algorithm uses a graph-based

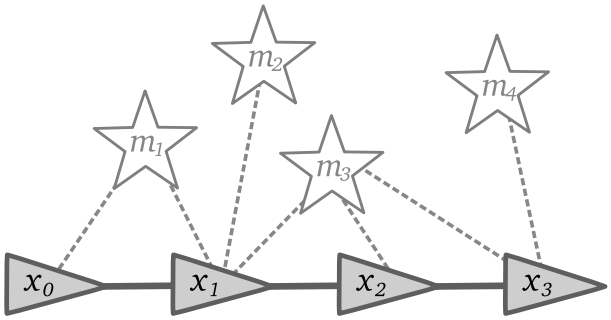


Figure 3 – Pose Graph.
Graph containing nodes to represent the robot's poses and features.

approach, where the robot poses that were sampled along a given trajectory and the features extracted from the environment are represented as nodes. The edges in the graph are built from the control, and also by connecting the poses of the features present in the environment to the poses of the robot at the exact moment when that feature was captured by a given sensor.

3.2.1 GraphSLAM Overview

Let $\boldsymbol{x} = x_{1:t} = [x_1, x_2, x_3, \cdots, x_t]^T$ be the robot's pose along it's trajectory, where $x_t \in \boldsymbol{x}$ is the robot's pose at time t. Let $\boldsymbol{z}^s = z^s_{1:t} = [z^s_1, z^s_2, z^s_3, \cdots, z^s_t]$ be the set of measurements from a given sensor s, where z^s_t is the measurement at time t from the sensor s. Let $\boldsymbol{u} = u_{1:t} = [u_1, u_2, u_3, \cdots, u_t]$ be the set of commands applied to the robot, where $u_t \in \boldsymbol{u}$ is the command applied at time t. Let $\boldsymbol{c}^s = c^s_{1:t} = [c^s_1, c^s_2, c^s_3, \cdots, c^s_t,]$ be the set of all correspondences between the measurements $\boldsymbol{z}^s$ and the features present in the map m, and $j = c^s_t$ is the correspondence that links z^s_t and a feature m_j on the map m.

For each measurement z_t, the algorithm creates a new node $\boldsymbol{N}_i$ in the graph to represent the robot's pose x_t at time t. In the same way, for each feature m_j related to one or more measurements, a new node is created to represent the pose of m_j on the map. Each command u_t aims to lead the vehicle from x_{t-1} to x_t, in this way, u_t imposes a constraint between these two subsequent poses. Finally, each measurement z_t that are related to a given feature m_j also generates a constraint between x_t and m_j, consequently, a new edge is added to connect the nodes related to x_t and m_j.

Fig. 3 shows the typical graph used by the GraphSLAM algorithm. $x_{0:4}$ are the robot's poses and $m_{1:4}$ are features extracted from the environment. The edges that connect the robot's poses are created from the commands $\boldsymbol{u}_{1:3}$. The dashed edges are links between the poses and the features. Section 4 explains how to build an extended version of this graph.

The GraphSLAM algorithm aims to find the best values for all nodes in such a way that it minimizes the sum of all constraints imposed by the edges. These constraints are

modeled as non-linear quadratic functions, therefore the minimization problem is solved as a non-linear least squares. However, the algorithm uses some procedures to reduce the complexity of the problem and solve it more efficiently.

Considering only the movement of the robot, a command u_t imposes a relative displacement between the poses x_{t-1} and x_t. This information can be used to define the following error function:

$$\boldsymbol{e}_t^o(\boldsymbol{x}, \boldsymbol{u}) = x_t - K(u_t, x_{t-1}) \tag{3.3}$$

Where $K(\cdot)$ is a kinematic function used as the robot's motion model. This function receives the previous pose x_{t-1} and the command u_t as inputs and then it outputs a new estimated pose x'_t. This error function is used to compose the sum of all related edges:

$$\boldsymbol{J}^o(\boldsymbol{x}, \boldsymbol{u}) = \sum_i^n (\boldsymbol{e}_i^o(\boldsymbol{x}, \boldsymbol{u}))^\intercal \hat{\boldsymbol{\Omega}}_i^o (\boldsymbol{e}_i^o(\boldsymbol{x}, \boldsymbol{u})) \tag{3.4}$$

where $\hat{\boldsymbol{\Omega}}_{\boldsymbol{i}}^{\boldsymbol{o}}$ is the odometry information matrix at node $\boldsymbol{N}_i$ projected through the non-linear function $K(.)$ via the unscented transformation [29].

The algorithm requires one node with a fixed pose. Usually, the first pose is defined as $x_0 = (0, 0, 0)^T$ and the information matrix is defined as the following diagonal matrix:

$$\hat{\Omega}_0^o = \begin{bmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{bmatrix} \tag{3.5}$$

The high values in the diagonal instruct the algorithm to not change the first node. Even small changes in the first node will increase the error 3.3.

The sensor measurements also create constraints between the nodes in the graph. Considering a sensor s and it's measurement z_t^s at time t, and the the robot's pose x_t, and the correspondence $j = c_t^s$ and a map m, one can compare z_t^s against the output of the inverse sensor model $h^s(x_t, m_j)$. This function estimates a measurement from the sensor s given x_t, the map m and the correspondence c_t^s. The measurement error is defined as the difference between the actual measurement z_t^s and the result from the inverse sensor model:

$$\boldsymbol{e}_t^s(\boldsymbol{x}, \boldsymbol{z}, m, \boldsymbol{c}) = z_t^s - h^s(x_t, m, c_t^s) \tag{3.6}$$

The error above can be defined for all sensors and all related constraints can be

joined in the following term:

$$\boldsymbol{J}^{\boldsymbol{S}}\left(\boldsymbol{x},\boldsymbol{z},m,\boldsymbol{c}\right)=\sum_{s\in\boldsymbol{S}}\sum_{t}(\boldsymbol{e}_{t}^{s}(\boldsymbol{x},\boldsymbol{z},m,\boldsymbol{c}))^{\intercal}\boldsymbol{\Omega}_{t}^{s}(\boldsymbol{e}_{t}^{s}(\boldsymbol{x},\boldsymbol{z},m,\boldsymbol{c})) \tag{3.7}$$

Where $\hat{\boldsymbol{\Omega}}_{\boldsymbol{t}}^{\boldsymbol{s}}$ is the information matrix of the measurement noise from sensor s and $\boldsymbol{S}$ is the set of all sensors.

Fig. 4 shows how the graph (at left) and the information matrix (at right) are updated given new measurements and movements. Fig 4(a) shows the initial context, where the robot is at the first pose x_1 and it detects a feature m_1. A new node containing m_1 is added to graph and the node is connected to the first node. The information matrix is updated to include the covariance between x_1 and m_1. Fig 4(b) shows the movement from x_1 to x_2. A new node is created in the graph and an new edge is created to connect x_1 and x_2. The information matrix contains now the covariance between x_1 and x_2. Fig. 4(c) shows the final result after several steps. One may notice that there are no links or edges between features

The global error function is defined as:

$$\begin{aligned}
J_{GraphSLAM} &= x_0^T\Omega_0^o x_0 \\
&+ \sum_t \left(x_t - K(u_t, x_{t-1})\right)^T \Omega_t^o (x_t - K(u_t, x_{t-1}) \\
&+ \sum_j \sum_{s\in\boldsymbol{S}} \sum_t (z_t^s - h^s(x_t, m_j))^T \Omega_t^s (z_t^s - h^s(x_t, m_j)) \\
&= x_0^T\Omega_0^o x_0 \\
&+ \sum_t (\boldsymbol{e}_t^o(\boldsymbol{x},\boldsymbol{u}))^{\intercal}\boldsymbol{\Omega}_t^o(\boldsymbol{e}_t^o(\boldsymbol{x},\boldsymbol{u})) \\
&+ \sum_{s\in\boldsymbol{S}}\sum_t (\boldsymbol{e}_t^s(\boldsymbol{x},\boldsymbol{z},m,\boldsymbol{c}))^{\intercal}\boldsymbol{\Omega}_t^s(\boldsymbol{e}_t^s(\boldsymbol{x},\boldsymbol{z},m,\boldsymbol{c})) \\
&= x_0^T\Omega_0^o x_0 + \boldsymbol{J}^{\boldsymbol{o}}(\boldsymbol{x},\boldsymbol{u}) + \boldsymbol{J}^{\boldsymbol{S}}(\boldsymbol{x},\boldsymbol{z},m,\boldsymbol{c})
\end{aligned} \tag{3.8}$$

The robot's poses and the map are found by minimizing this global error function.

3.2.2 Mathematical Derivation of GraphSLAM

This section presents the mathematical derivation of the GraphSLAM algorithm while using the notation adopted in this work. One can group the initial pose $x_0 = (0,0,0)^T$, the unknown poses $x_{1:t}$, and the map m to compose the state vector $y_{0:t}$:

$$y_{0:t} = \begin{pmatrix} x_0 & x_1 & \cdots & x_t & m \end{pmatrix}^T \quad and \quad y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{3.9}$$

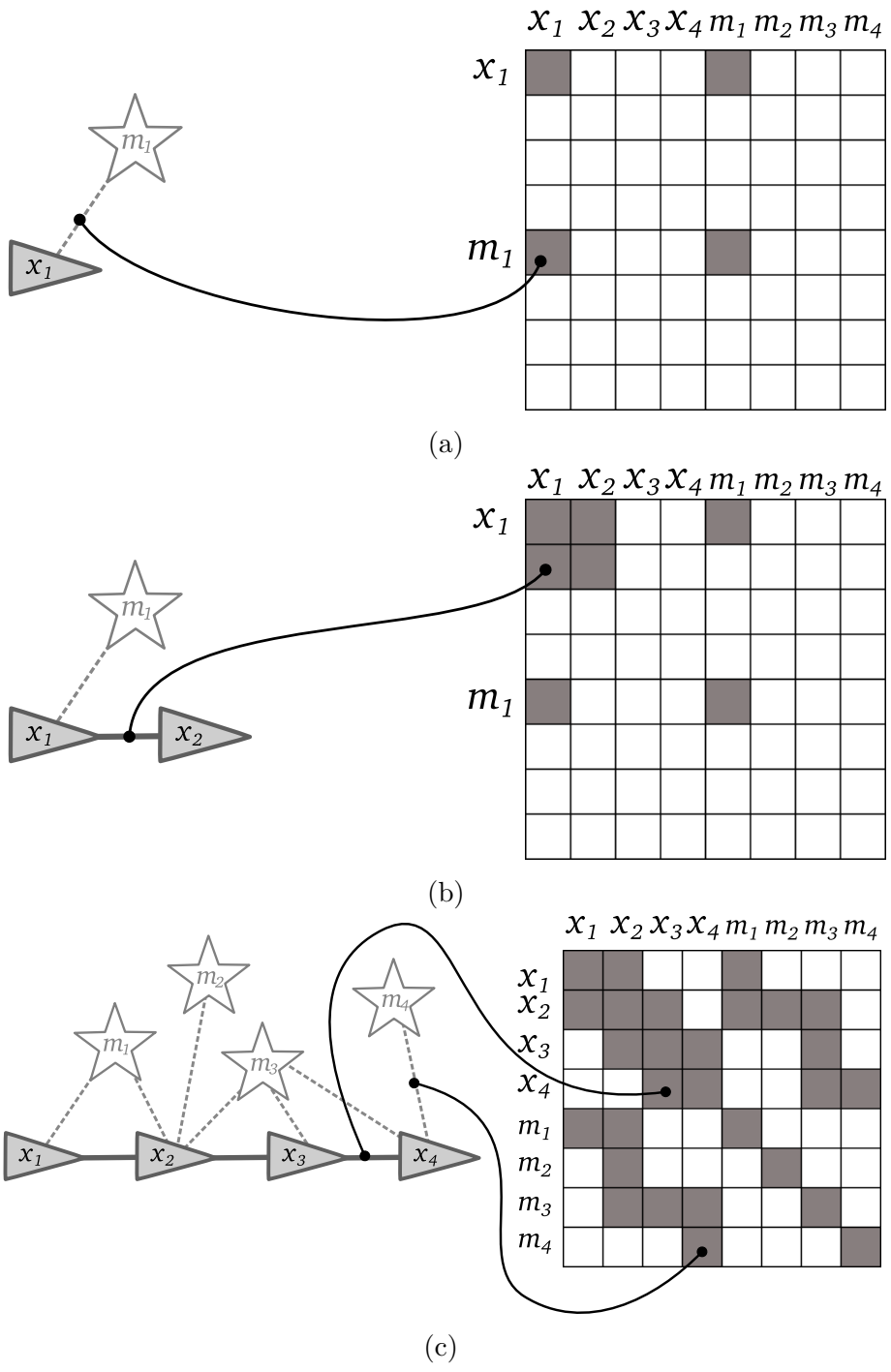


Figure 4 – Graph formation.
Graph formation. (a) A feature is observed by the robot. A new node and an edge are added to the graph. (b) A node and an odometry edge are added to graph when the robot moves from x_1 to x_2. (c) The graph and information matrix updates after several measurements.

The algorithm assumes that the sensors noise and the errors in the robot's movement have a Gaussian distribution. Therefore, all properties of the Gaussian distribution can be used to facilitate the mathematical derivation and to solve the problem. For example, consider $K(u_t, x_{t-1})$ as the deterministic function that receives the command u_t and the

previous pose x_{t-1}, and then estimates a new pose $x_t^{'}$. Let R_t be the covariance matrix of the error in the robot's movements. The related Gaussian distribution is defined as:

$$\mathcal{N}(K(u_t, x_t), R_t) \tag{3.10}$$

Similarly, considering $h^s(y_t, c_t^s)$ as a function that receives the pose x_t and the map m (both inside y_t), and the correspondence $j = c_t^s$ as input and then it outputs the expected measurement from the sensor s. Let Q_t be the covariance matrix of the error in the measurements of the sensor s. The related Gaussian distribution is defined as:

$$\mathcal{N}(h^s(y_t, c_t^s), Q_t) \tag{3.11}$$

The posterior probability of the full SLAM problem can be defined as:

$$p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) \tag{3.12}$$

The equation (3.12) can be read as the joint probability of all poses $x_{0:t}$ and the map m given the the sensor measurements $z_{1:t}$, the commands $u_{1:t}$ and the correspondences $c_{1:t}$. This equation can be manipulated through the Bayes rule to separate the probability of the last measurement z_t:

$$\begin{aligned} &p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = \\ &\qquad \eta\, p(z_t|y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t})\; p(y_{0:t}|z_{1:t-1}, u_{1:t}, c_{1:t}) \end{aligned} \tag{3.13}$$

Where η is the usual constant found in the Gaussian distributions. One may notice that each probability on the right side of (3.13) have conditional variables that do not interfere in the resulting probabilities. The first probability does not depend on the commands $u_{1:t}$ or the previous measurements $z_{1:t-1}$. Instead, the probability depends on the last pose and the map (both inside y_t) and on the current correspondence c_t. Consequently, the term can be written as:

$$p(z_t|y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) = p(z_t|y_t, c_t) \tag{3.14}$$

The later probability on the right side of the equation (3.12) can factored by separating the last pose from the previous ones:

$$
\begin{aligned}
&p(y_{0:t}|z_{1:t-1}, u_{1:t}, c_{1:t}) \\
&\qquad = p(x_t|y_{0:t-1}, z_{1:t-1}, u_{1:t}, c_{1:t}) p(y_{0:t-1}|z_{1:t-1}, u_{1:t}, c_{1:t}) \\
&\qquad = p(x_t|x_{n-1}, u_t) p(y_{0:t-1}|z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \qquad (3.15)
\end{aligned}
$$

Applying (3.14) and (3.15) into (3.12), one can obtain the recursive definition of the full SLAM problem:

$$
\begin{aligned}
&p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = \\
&\qquad \eta\, p(z_t|y_t, c_t)\; p(x_t|x_{n-1}, u_t)\; p(y_{0:t-1}|z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \qquad (3.16)
\end{aligned}
$$

The recursion above can be seen as a recurrence relation and it can be solved by known methods, such as generating functions. The solution of this recurrence relation leads to the formula:

$$
\begin{aligned}
&p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = \\
&\qquad \eta\, p(x_0) \prod_{s \in \boldsymbol{S}} \left(\prod_{t=1:n} p(z_t^s|y_t, c_t^s) \right) \qquad (3.17)
\end{aligned}
$$

Since the formula above contains multiplication of probabilities, the result tends quickly to zero or one due to the limited precision. Hence, the full SLAM problem is reduced to a dual optimization after applying the *log* function on both sides. The *log* function transforms the multiplication on (3.17) into summation.

$$
\begin{aligned}
&log\; p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) \\
&\qquad = const. + log\; p(x_0) \\
&\qquad + \sum_t \left[log\; p(x_t|x_{t-1}, u_t) + \sum_{s \in \boldsymbol{S}} log\; p(z_t^s|y_t, c_t^s)) \right] \qquad (3.18)
\end{aligned}
$$

The initial pose x_0 is fixed at $x_0 = (0, 0, 0)^T$, hence the probability $p(x_0)$ on (3.18) can be defined by a Gaussian distribution:

$$
p(x_0) = \eta\; e^{-\frac{1}{2}((x_0)^T R_0^{-1} x_0} \qquad (3.19)
$$

Where R_0 is usually the covariance matrix of the error in the robot's movements. As indicated in (3.5), the final information matrix (inverse of the covariance matrix) is conveniently set to a diagonal matrix with high values in its diagonal.

The term $p(x_t|x_{t-1}, u_t)$ on (3.18) is the probability of the pose x_t given the previous pose x_{t-1} and the command u_t applied to the robot. Assuming that this translation has a Gaussian error, one can use (3.10) to build the following definition:

$$\begin{aligned} p(x_t|x_{t-1}, u_t) &= \eta\, e^{-\frac{1}{2}\left((x_t - K(u_t, x_{t-1}))^T R_t^{-1} (x_t - K(u_t, x_{t-1}))\right)} \\ &= \eta\, e^{-\frac{1}{2}\left((e_t^o(\boldsymbol{x}, \boldsymbol{u}))^T R_t^{-1} (e_t^o(\boldsymbol{x}, \boldsymbol{u}))\right)} \end{aligned} \tag{3.20}$$

The term $p(z_t^s|y_t, c_t^s)$ on (3.18) is the probability of the measurement z_t^s at time t given the pose x_t, the map m and the correspondence $j = c_t^s$. Assuming the Gaussian distribution to model the errors in the measurements, then the probability can be defined in the form:

$$\begin{aligned} p(z_t^s|y_t, c_t^s) &= \eta\, e^{-\frac{1}{2}(z_t^s - h^s(y_t, c_t^s))^T Q_t^{-1} (z_t^s - h^s(y_t, c_t^s))} \\ &= \eta\, e^{-\frac{1}{2}(\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}))^T Q_t^{-1} (\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}))} \end{aligned} \tag{3.21}$$

Here we have the inverse sensor model with different parameters. One should remember that (3.6) has defined the same function in this format $h^s(x_t, m, c_t^s)$. Since we can use the $y_{0:t}$ vector, which contains the poses $\boldsymbol{x}$ and the map m, and the features m_j can be recovered from the map and the correspondences $\boldsymbol{c}$, then we have a more compact notation:

$$\boldsymbol{e}_t^s(\boldsymbol{x}, \boldsymbol{z}, m, \boldsymbol{c}) = \boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}) \tag{3.22}$$

Applying (3.19), (3.20) and (3.21) into (3.18) results in:

$$\begin{aligned} log\; p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) & \\ &= const. + log\left(\eta\, e^{-\frac{1}{2}((x_0)^T R_0^{-1} x_0)}\right) \\ &+ \sum_t \left[log\left(\eta\, e^{-\frac{1}{2}(\boldsymbol{e}_t^o(\boldsymbol{x}, \boldsymbol{u}))^\intercal \boldsymbol{R}_t^{-1} (\boldsymbol{e}_t^o(\boldsymbol{x}, \boldsymbol{u}))}\right)\right. \\ &\left. + \sum_{s \in S} log\left(\eta\, e^{-\frac{1}{2}(\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}))^\intercal \boldsymbol{Q}_t^{-1} (\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}))}\right)\right] \end{aligned} \tag{3.23}$$

The next procedure consists of taking advantage of the relations between the logarithmic and the exponential and also multiplying both sides by -1, and finally grouping the constant terms:

$$
\begin{aligned}
-log\ p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = const. \\
&+ \frac{1}{2}\Bigg[(x_0)^T R_0^{-1} + \sum_t (\boldsymbol{e}_t^o(\boldsymbol{x}, \boldsymbol{u}))^\intercal \boldsymbol{R}_t^{-1} (\boldsymbol{e}_t^o(\boldsymbol{x}, \boldsymbol{u})) \\
&+ \sum_{s \in \boldsymbol{S}} \sum_t (\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}))^\intercal \boldsymbol{Q}_t^{-1} (\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}))\Bigg]
\end{aligned} \tag{3.24}
$$

Beside the constants, one can see that the equation (3.24) is similar to (3.8).

All constraints defined so far are quadratic and they depend on the $K(\cdot)$ and $h^s(\cdot)$ functions. These two functions are non-linear, therefore, the next steps require the linearization of these functions by expanding them through a Taylor series. The linearization guarantees that the constraints will be directly dependent on the variables to be estimated ($x_{1:t}$ and m). [30] presents the Taylor expansions up to the first derivative:

$$
K(u_t, x_{t-1}) \cong K(u_t, \mu_{t-1}) + \underbrace{K^{'}(u_t, \mu_{t-1})}_{\text{G}}(x_{t-1} - \mu_{t-1}) \tag{3.25}
$$

$$
h^s(y_t, c_t^s) \cong h^s(\mu_t, c_t^s) + \underbrace{h^{s^{'}}(\mu_t, c_t^s)}_{\text{H}}(y_t - \mu_t) \tag{3.26}
$$

Where the term μ_t is the current estimate of the state vector y_t. The term G is the Jacobian obtained from the derivatives of the function $K(\cdot)$ with respect to y_{t-1} evaluated at u_t and μ_{t-1}. The term H is the derivative of $h^s(\cdot)$ with respect to y_t. Applying (3.25) to (3.3) and also (3.26) to (3.6), on can obtain the following definitions:

$$
\boldsymbol{e}_t^o(\boldsymbol{x}, \boldsymbol{u}) \cong x_t - K(u_t, \mu_{t-1}) + G(x_{t-1} - \mu_{t-1}) \tag{3.27}
$$

$$
\boldsymbol{e}_t^s(\boldsymbol{y}, \boldsymbol{c}) \cong z_t^s - h^s(\mu_t, c_t^s) + H(y_t - \mu_t) \tag{3.28}
$$

Now, one can use the results from (3.27) and (3.28) to update (3.24):

$$
\begin{aligned}
-log\ p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = const. \\
&+ \frac{1}{2}\Bigg[(x_0)^T R_0^{-1}(x_0) \\
&+ \sum_t (x_t - K(u_t, \mu_{t-1}) + G(x_{t-1} - \mu_{t-1}))^T R_t^{-1} (x_t - K(u_t, \mu_{t-1}) \\
&\quad + G(x_{t-1} - \mu_{t-1})) \\
&+ \sum_{s \in \boldsymbol{S}} \sum_t (z_t^s - h^s(\mu_t, c_t^s) + H(y_t - \mu_t))^T Q_t^{-1} (z_t^s - h^s(\mu_t, c_t^s) + H(y_t - \mu_t))\Bigg]
\end{aligned} \tag{3.29}
$$

After the linearization, the terms became quadratic with respect to $y_{0:t}$. The equation above can be re-organized and some terms can be grouped for more clarity:

$$
\begin{aligned}
-log\ p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) &= const. \\
&+ \frac{1}{2} \underbrace{(x_0)^T R_0^{-1}(x_0)}_{\text{quadratic on the first pose}} \\
&+ \frac{1}{2} \sum_t \underbrace{(x_{t-1:t})^T \begin{pmatrix} 1 \\ -G \end{pmatrix} R_t^{-1}(1-G)(x_{t-1:t})}_{\text{quadratic on last two poses}} \\
&\quad + \underbrace{(x_{t-1:t})^T \begin{pmatrix} 1 \\ -G \end{pmatrix} R_t^{-1}[K(u_t.\mu_{t-1}) + G\mu_{t-1}]}_{\text{linear on the last two poses}} \\
&+ \frac{1}{2} \underbrace{(y_t)^T H^T Q_t^{-1} H(y_t)}_{\text{quadratic on the state vector y}} + \underbrace{((y_t)^T H^T Q_t^{-1}[z_t^s - h^s(\mu_t, c_t^s) - H\mu_t])}_{\text{linear on the state vector y}}
\end{aligned}
\tag{3.30}
$$

Now, all quadratic terms should be moved to a matrix $\mathbf{\Omega}$ and all linear terms should be placed in a vector $\boldsymbol{\xi}$:

$$
-log\ p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = const. + \frac{1}{2}(y_{0:t})^T \mathbf{\Omega}(y_{0:t}) + (y_{0:t})^T \boldsymbol{\xi} \tag{3.31}
$$

The next step multiplies (3.31) by -1 and applies the exponential function on both sides:

$$
p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = const. + e^{\frac{1}{2}(y_{0:t})^T \mathbf{\Omega}(y_{0:t}) + (y_{0:t})^T \boldsymbol{\xi}} \tag{3.32}
$$

The right side of the equation (3.32) represents a Gaussian distribution in its information form, hence one can infer the following:

$$
p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = \mathcal{N}(\mathbf{\Omega}, \boldsymbol{\xi}) \tag{3.33}
$$

The GraphSLAM algorithm can recover the path and the map from $\mathbf{\Omega}$ and $\boldsymbol{\xi}$:

$$
\Sigma = \mathbf{\Omega}^{-1} \tag{3.34}
$$

$$
\mu = \Sigma\xi \tag{3.35}
$$

Where μ is the mean of the Gaussian distribution that describes the poses and the map.

The solution for the system of equations above can be easy when there are no loop closures. In this case $\mathbf{\Omega}$ would be a band matrix with efficient inversion. The solution would be in linear time [30]. However, the typical cases are those with loop closures. The robots usually pass by the same places multiple times and then the same feature can be detected twice or more. The algorithm adopts a procedure to remove all edges that are related to features. In other words, the algorithm removes all nodes that represent the poses of the features in the map.

When a feature is linked to a single pose, then the edge is removed without any additional procedure. When a feature is linked to multiple poses, then the algorithm creates a new edge connecting each pair of poses that are linked to the same feature. This approach translates all information between poses and features to information between poses. Therefore, the problem is restricted to only find the path of the robot. One can consider that if a feature m_j has a certain distance from a pose x_i and a distance to a pose x_j, then one can infer the distance between x_i and x_j if the angles are known.

This procedure can be better understood through a clear mathematical description. The first step is to separate the probabilities of the poses and the map on equation (3.12):

$$p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = \\ \eta\, p(x_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) p(m|x_0, z_{1:t}, u_{1:t}, c_{1:t}) \tag{3.36}$$

The first part of the algorithm tries to find only the poses given the measurements, commands and the correspondences between features and measurements. The first probability on the right side of (3.36) still have a Gaussian distribution, so the following is valid:

$$p(x_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) \cong \mathcal{N}(\widetilde{\mathbf{\Omega}}, \widetilde{\boldsymbol{\xi}}) \tag{3.37}$$

$\widetilde{\mathbf{\Omega}}$ and $\widetilde{\boldsymbol{\xi}}$ can be found from $\mathbf{\Omega}$ and $\boldsymbol{\xi}$ by first executing the partitioning below:

$$\mathbf{\Omega} = \begin{pmatrix} \Omega_{x_{0:t},x_{0:t}} & \Omega_{x_{0:t},m} \\ \Omega_{m,x_{0:t}} & \Omega_{m,m} \end{pmatrix} \tag{3.38}$$

$$\boldsymbol{\xi} = \begin{pmatrix} \xi_{x_{0:t}} \\ \xi_m \end{pmatrix} \tag{3.39}$$

Fig. 5 shows the information matrix $\mathbf{\Omega}$ and the resulting partition. The part related to $\Omega_{m,m}$ is a block diagonal matrix since there are no feature-feature relations.

From the *marginalization lemma* [28], $\widetilde{\mathbf{\Omega}}$ and $\widetilde{\boldsymbol{\xi}}$ can be found by the formulas below:

$$\widetilde{\mathbf{\Omega}} = \Omega_{x_{0:t},x_{0:t}} - \Omega_{x_{0:t},m}\Omega_{m,m}^{-1}\Omega_{m,x_{0:t}} \tag{3.40}$$

$$\widetilde{\boldsymbol{\xi}} = \xi_{x_{0:t}} - \Omega_{x_{0:t},m}\Omega_{m,m}^{-1}\xi_m \tag{3.41}$$

Apparently, the inversion of $\Omega_{m,m}$ should be a hard task but there are efficient solution for block diagonal matrices. Once $\widetilde{\mathbf{\Omega}}$ and $\widetilde{\boldsymbol{\xi}}$ are found, one can obtain $\widetilde{\mu}$ through the system below:

$$\widetilde{\Sigma} = \widetilde{\mathbf{\Omega}}^{-1} \tag{3.42}$$

$$\widetilde{\mu} = \widetilde{\Sigma}\widetilde{\boldsymbol{\xi}} \tag{3.43}$$

Where $\widetilde{\mu}$ is the mean of a Gaussian distribution that describes the poses of the robot.

After finding the poses along the entire trajectory, the algorithm needs to solve the probability $p(m|x_{0:t}, u_{1:t}, c_{1:t})$ on the equation (3.36). The *conditioning lemma* [28] is used in (3.38) and (3.39) to demonstrate that the probability of the map still follows a Gaussian distribution. The parameters of the Gaussian distribution that describes the map are obtained by the following formulas:

$$\Sigma_m = \Omega_{m,m}^{-1} \tag{3.44}$$

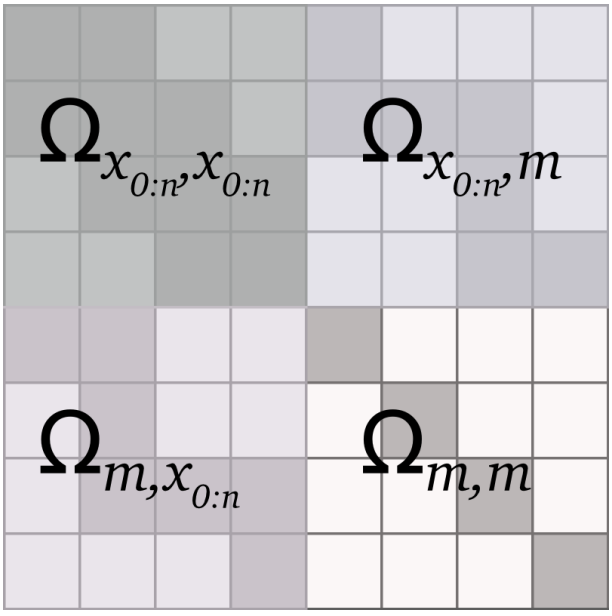


Figure 5 – Information matrix.
The information matrix is partitioned in four matrices.

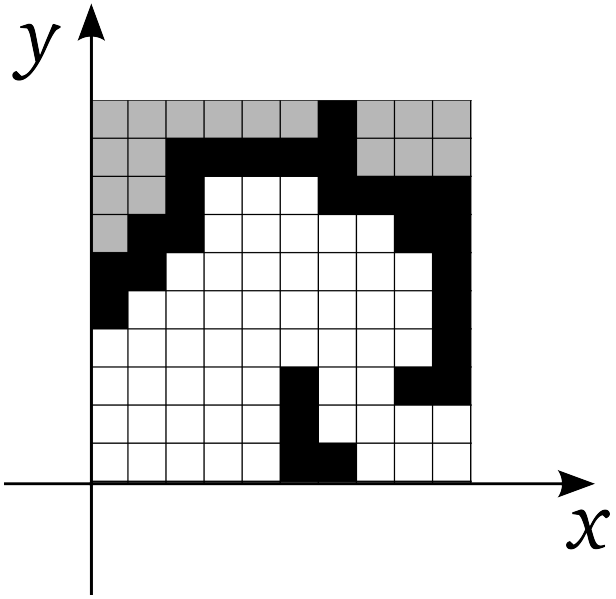


Figure 6 – The occupancy grid map.
In this example, white cells have a low probability of being occupied, black cells have a high probability of being occupied and gray cells are unknown.

$$\mu_m = \Sigma_m(\xi_m + \Omega_{m,x_{0:t}}\widetilde{\xi}) \tag{3.45}$$

Where μ_m is the mean of the Gaussian distribution that describes the map. This value depends on $\widetilde{\boldsymbol{\xi}}$ computed in the previous step. Once $\widetilde{\boldsymbol{\xi}}$ and μ_m are found, the full SLAM problem is completely solved.

3.3 Occupancy Grid Mapping

The occupancy grid map (OGM) is a representation of the environment through a grid of cells with the same dimensions. Fig. 6 exhibits the basic schemed adopted by the OGM. One may notice that this representation is discrete and with finite resolution. Each cell contains a probability P of being occupied given the current pose x_t and the current measurement z_t. This probability P is also known as the inverse model $p(m_i|x_t, z_t)$.

P is always a value between 0 and 1, hence the OGM uses some threshold values to decide the cell's state. For example, one may consider that cells containing values greater than 0.75 are occupied, and cells containing values lesser than 0.25 are empty, and cells with any value between 0.25 and 0.75 have an uncertain state. This probability is also known as the inverse model.

The OGMs are built by measuring the environment and projecting the measurements to the grid map. This projection requires the correct pose of the sensor at each measurement. Initially, all cells are set to an uncertain state (e.g. $P = 0.5$). The OGM iteratively updates the probabilities at each cell that is reached by the current measurement.

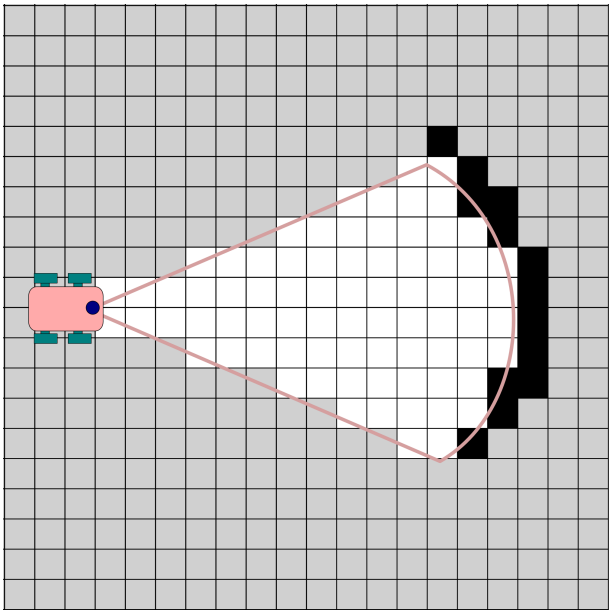


Figure 7 – Mapping.

Fig. 7 shows the update of some cells that are reached by the sensor. If the sensor detects an obstacle, the projection of this obstacle to the grid map increases the probability of the related cells. On the other hand, the space between the sensor and the obstacle decreases the probability of the related cells.

The OGM uses the *log-odds* representation to avoid numerical instabilities and to facilitate the update of the probabilities on each cell. So, instead of storing the raw probabilities, each cell contains:

$$L = log\left(\frac{P}{1-P}\right) = inverse\_sensor\_model(m_i, x_t, z_t) \tag{3.46}$$

The $inverse\_sensor\_model(\cdot)$ is a function that calculates the inverse model $p(m_i|x_t, z_t)$ in it's *log-odds* format.

When needed, the probability P can be recovered from the *log-odds*:

$$P = 1 - \frac{1}{1+e^L} \tag{3.47}$$

Let l_{0,m_i} be the *log-odds* of cell m_i in the map m at time $t = 0$. The update of the m_i is done by:

$$l_{t,m_i} = l_{t,m_{i-1}} + inverse\_sensor\_model(m_i, x_t, z_t) - l_{0,m_i} \tag{3.48}$$

4 Large-scale Mapping System

In this section, we describe the proposed *full* SLAM-based mapping system, including details about its *front-end* and *back-end*. The *front-end* consists of the Pre-processing and the Hypergraph Builder modules, and it is responsible for handling the low level information coming from the sensors, converting between data formats, filtering and building the hypergraph for optimization. The *back-end* consists of the Hypergraph Optimizer and the Mapper modules, and it is responsible for estimating the AV poses along the multiple sessions, calibrating the odometry bias and building and merging the local grid maps. In the remaining of the text, we refer directly to these modules.

The proposed mapping system is capable of building many types of grid maps, such as the occupancy grid maps (OGM), remission maps (RM) and likelihood field maps (LFM). However, the AV for which it was designed uses mainly occupancy grid maps for navigation and localization. Therefore, the OGM will be focused in the next sections.

4.1 System Overview

Fig. 8 shows the architecture of the proposed mapping system. The AV is first driven by a human through the environment while sensors data are recorded in log files. The sensor measurements are saved as they arrive in the logging tool. The Pre-processing module reads the input logs, converts the raw sensor messages to appropriate formats, and applies a filtering scheme to remove noisy and needless information. Then, the Hypergraph Builder module creates a graph containing nodes to represent the unknown AV poses, nodes to represent the odometry bias and many types of edges to represent the constraints between the nodes. The constraints are defined by the sensor data and each sensor type generates different types of edges or constraints. After that, the Hypergraph Optimizer module finds the most likely AV poses and odometry bias calibration given all restrictions imposed by the edges in the hypergraph. Finally, the Mapper module takes the final AV poses and the input logs to create and merge the grid maps.

4.2 Pre-processing

Sensors data come from parallel processes running in one or more computers. Hence, they may be stored in an order different from that observed during capture. In the first step, the Pre-Processing module sorts all measurements by their timestamps. In the second step, it discards some sensor measurements when the AV is below a minimum speed $MinSpeed$. Some sensor measurements do not contribute positively for the mapping process when the

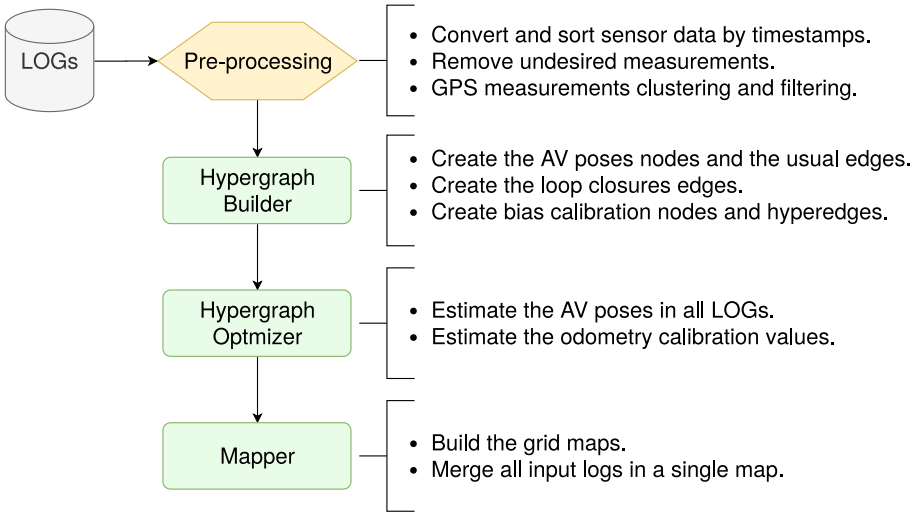


Figure 8 – System overview.

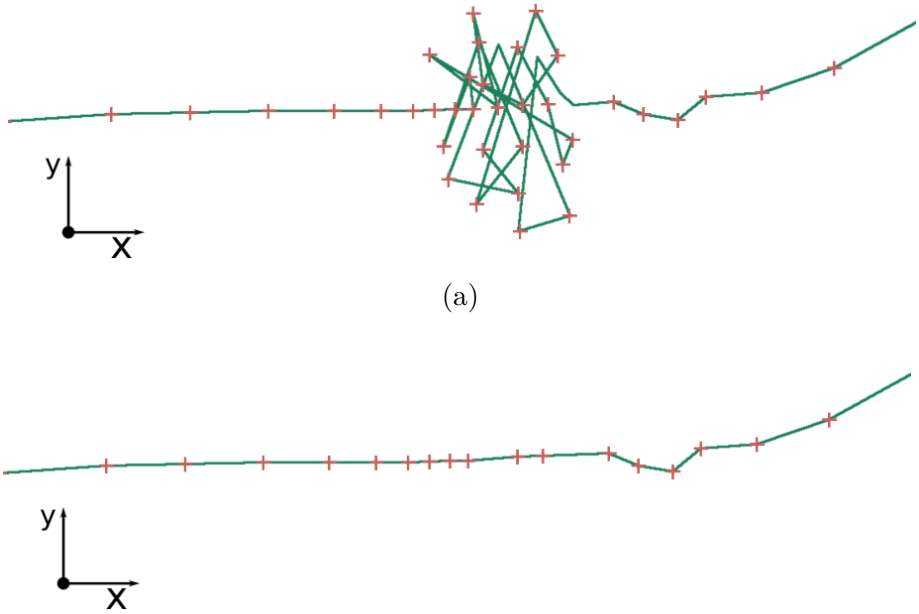


Figure 9 – GPS noise: the speed filtering.
(a) GPS noise at low speed. (b) GPS noise removal after filtering.

AV is stopped or at very low speeds. Instead, they tend to incur errors. Fig. 9(a) shows noisy GPS readings when the vehicle is stopped and Fig. 9(b) illustrates the result after speed filtering.

In the case of laser scans, in an ideal world with a perfect LiDAR sensor and an environment with only static objects, the resulting point clouds at zero speed should always present the same points, so that the redundant point clouds can be safely discarded as they do not provide any extra information. However, in the real world, the LiDAR sensor has inevitable noise and the environment may have moving objects, but we still

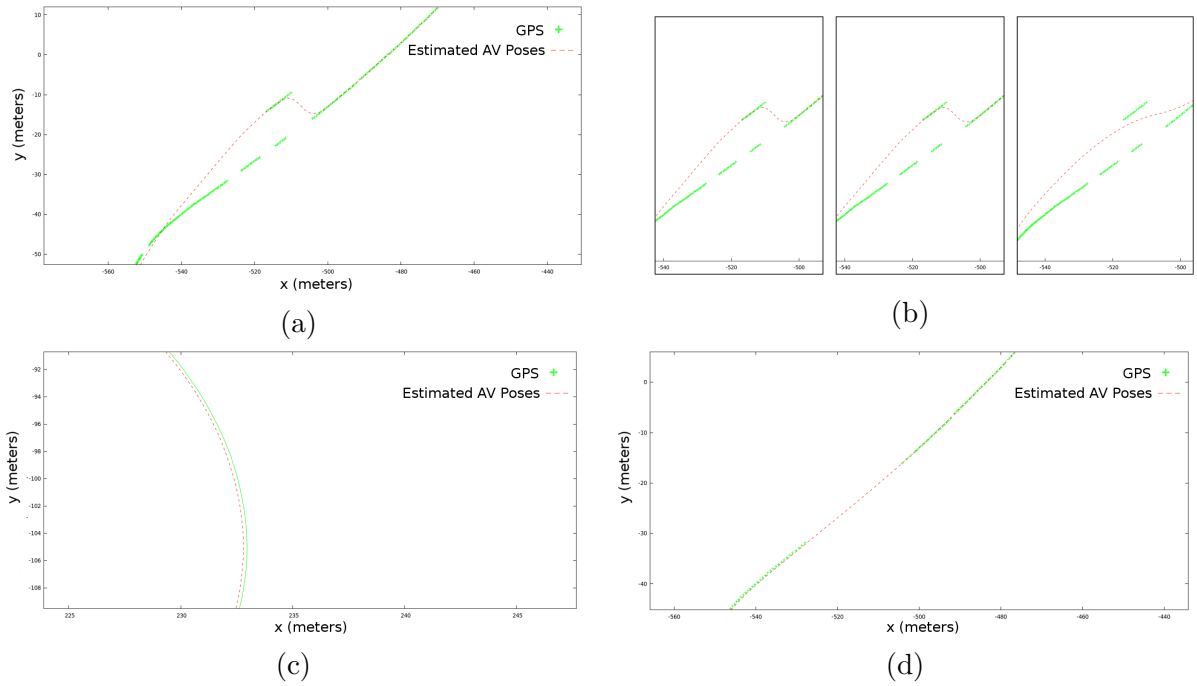


Figure 10 – GPS lateral displacement error.
(a) GPS lateral displacement error. (b) Increasing the GPS error variance from left to right. (c) Undesired side effects in the estimated AV poses after increasing the variance of GPS error. (d) Estimated AV poses after clustering the GPS measurements and removing small groups.

can discard point clouds given moving objects are undesired and static objects will be eventually seen by the LiDAR sensor when the AV moves again.

In the final step, the Pre-Processing module removes GPS lateral displacement errors. GPS may suffer a lateral displacement error, as illustrated in Fig. 10(a). The probabilistic treatment to this error [28] would require a lower confidence in the GPS measurement; in other words, assuming a normal distribution to model the GPS error we should increase its variance parameter. Fig. 10(b) shows a sequence of increasing GPS error variances and the resulting effects in the AV poses. Although the displacement error in the AV poses decreases in that particular spot, the estimated path moves away from the GPS measurements in other areas. Fig. 10(c) illustrates undesired side effects in another region with better GPS measurements.

Our solution to the GPS lateral displacement error considers that other sensors provide good local positioning and we discard the GPS data in the critical areas. The GPS measurements are clustered based on the distance between immediate neighbors $NeighborDistance$. A divisive hierarchical clustering can be applied to properly separate unwanted groups. Initially, all measurements can be considered as a single cluster and, for any pair of immediate neighbors, $\boldsymbol{z}_i^{GPS}$ and $\boldsymbol{z}_{i+1}^{GPS}$, if the distance between the neighbors is greater than $NeighborDistance$, then we can split the initial cluster and recursively apply the method in the new smaller clusters. After clustering, the method removes all groups

containing fewer elements than a desired quantity $NeighborQuantity$. Fig. 10(d) presents the final AV poses after removing the error.

4.3 Hypergraph Building

After the pre-processing, we consider the resulting sensor data as valid and we use them to build the hypergraph. The nodes in the hypergraph represent the unknown AV poses and the odometry biases. Edges represent constraints between the nodes.

A pose node $\boldsymbol{N}_i$ is added to the hypergraph for each sensor measurement $\boldsymbol{z}_i$. $\boldsymbol{N}_i$ refers to an unknown pose $\boldsymbol{x}_i$:

$$\boldsymbol{N}_i = \boldsymbol{x}_i = (x_i, y_i, \theta_i) \tag{4.1}$$

where (x_i, y_i) is the position of the vehicle in the ground plane and θ_i is its heading.

One may notice that we changed the subscript t to i. This is done to accommodate nodes that are not related to the poses. The calibration nodes are not related to time.

4.3.1 Odometry Edges

For each log, the Hypergraph Builder traverse the nodes in the associated hypergraph considering each node pair N_i and N_{i+1} to build the odometry edges. The last known odometry measurement is used to create an edge connecting the two nodes. More precisely, the last odometry measurement contains the ego vehicle velocity v_i and steering angle φ_i used to define the current command $\boldsymbol{u}_i$ alongside Δ_t, which is the time difference between the two nodes. The odometry edge indicates a constraint between $\boldsymbol{N}_i$ and $\boldsymbol{N}_{i+1}$ through the error function $\boldsymbol{e}^o(\boldsymbol{x}_i, \boldsymbol{u}_i)$.

We simplify the notation by encoding the involved quantities in the indices of the error function [31]:

$$\boldsymbol{e}^o(\boldsymbol{x}_i, \boldsymbol{u}_i) \overset{def.}{=} \boldsymbol{e}^o_i(\boldsymbol{x}, \boldsymbol{u}) \tag{4.2}$$

The odometry constraint between each node pair is defined as:

$$\boldsymbol{e}^o_i(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x}_{i+1} \ominus \boldsymbol{x}_i) \ominus K(\boldsymbol{u}_i, \Delta t) \tag{4.3}$$

where $\ominus$ is the inverse of the motion composition operator $\oplus$ as defined in [32], Δt is the time difference between the poses $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ and $K(.)$ is a forward kinematic function that returns a relative movement given $\boldsymbol{u}_i$ and Δt. The function $K(.)$ uses the Ackerman

steering geometry with understeer corrections [33]. All odometry edges are combined in the total odometry error:

$$\boldsymbol{J}^{o}\left(\boldsymbol{x},\boldsymbol{u}\right)=\sum_{i}^{n}(\boldsymbol{e}_{i}^{o}(\boldsymbol{x},\boldsymbol{u}^{\intercal}))^{\intercal}\hat{\boldsymbol{\Omega}}_{i}^{o}(\boldsymbol{e}_{i}^{o}(\boldsymbol{x},\boldsymbol{u}^{\intercal})) \tag{4.4}$$

where $\hat{\Omega}_{i}^{o}$ is the odometry information matrix at node $\boldsymbol{N}_i$ projected through the non-linear function $K(.)$ via the unscented transformation [29].

4.3.2 GPS Edges

Each node $\boldsymbol{N}_i$ created by a GPS measurement $\boldsymbol{z}_{i}^{GPS}$ receives a unary edge containing the error:

$$\boldsymbol{e}_{i}^{GPS}\left(\boldsymbol{x}\right)=\boldsymbol{x}_{i}\ominus\boldsymbol{z}_{i}^{GPS} \tag{4.5}$$

All GPS edges constraints are combined in the total GPS error:

$$\boldsymbol{J}^{GPS}\left(\boldsymbol{x}\right)=\sum_{i}(\boldsymbol{e}_{i}^{GPS}(\boldsymbol{x}))^{\intercal}\boldsymbol{\Omega}_{i}^{GPS}(\boldsymbol{e}_{i}^{GPS}(\boldsymbol{x})) \tag{4.6}$$

where Ω_{i}^{GPS} is the inverse of the GPS covariance matrix at node $\boldsymbol{N}_i$.

4.3.3 Loop Closure Edges

We use LiDAR and GPS sensors data to compute loop closures in the same log (intra-session) and also loop closures between different logs (inter-session). Let $\boldsymbol{N}^{L}$ be the subset of all nodes generated by the LiDAR sensor and $\boldsymbol{N}^{GPS}$ the subset of all nodes generated by the GPS sensor. Given a node $\boldsymbol{N}_i \in \boldsymbol{N}^{L}$, we search for the nearest node $\boldsymbol{N}_j \in \boldsymbol{N}^{L}$ with time difference Δt greater than a desired threshold $InterLoopTime$ (e.g., 1 minute) and also distance less than another threshold value $LoopDistance$ (e.g., 5 meters). We use the GPS measurements to estimate the distances between $\boldsymbol{N}_i$ and the other nodes in $\boldsymbol{N}^{L}$. Let $\boldsymbol{N}_k \in \boldsymbol{N}^{GPS}$ be the node with the closest timestamp to $\boldsymbol{N}_i$ and $\boldsymbol{N}_l \in \boldsymbol{N}^{GPS}$ the node with the closest timestamp to $\boldsymbol{N}_j$. We assume that the distance between $\boldsymbol{N}_i$ and $\boldsymbol{N}_j$ is equal to the distance between $\boldsymbol{N}_k$ and $\boldsymbol{N}_l$.

For every pair of poses $\boldsymbol{N}_i$ and $\boldsymbol{N}_j$ that meet both conditions (minimum time and maximum distance thresholds), we add a new loop closure edge. This method is applied to find inter-session and intra-session loop closures. It is important to mention that the inter-session loop closures are vital to connect the different logs in the global hypergraph.

The inter-session loop closure edges provide required relationships between AV poses in multiple logs. A loop closure edge computes the following error as defined in [31]:

$$\boldsymbol{e}_{i,j}^{L}(\boldsymbol{x}) = (\boldsymbol{x}_j \oplus \boldsymbol{s}^{L}) \ominus (\boldsymbol{x}_i \oplus \boldsymbol{s}^{L}) \ominus ICP_{i,j}^{L} \tag{4.7}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the AV poses represented by the nodes $\boldsymbol{N}_i$ and $\boldsymbol{N}_j$, $\boldsymbol{s}^{L}$ is the pose of the LiDAR sensor in the AV reference frame and $ICP_{i,j}^{L}$ is a linear transform that aligns the point cloud $\boldsymbol{z}_i^{L}$ with the point cloud $\boldsymbol{z}_j^{L}$. We use the Generalized Iterative Closest Point (GICP) [34] algorithm to find the transform. The error below combines the constraints of all loop closure edges:

$$\boldsymbol{J}^{L}(\boldsymbol{x}) = \sum_{i} (\boldsymbol{e}_{i,j}^{L}(\boldsymbol{x}))^{\intercal} \boldsymbol{\Omega}_{i,j}^{L} (\boldsymbol{e}_{i,j}^{L}(\boldsymbol{x})) \tag{4.8}$$

where Ω_i^{L} is the inverse of the GICP covariance matrix.

4.3.4 Odometry Calibration Edges

Mutz et al. [3] showed that odometry measurements are subject to biases in the velocity v and steering wheel angle φ, which introduce even more error in dead reckoning. They then proposed an odometry bias calibration method before optimizing the graph, which makes odometry measurements more reliable and allows the odometry variance parameter to be decreased in a pose graph SLAM algorithm. The odometry bias calibration is approximated by the following linear functions:

$$v_i^{'} = v_i b_{mult}^{v} \tag{4.9}$$

$$\varphi_i^{'} = \varphi_i b_{mult}^{\varphi} + b_{add}^{\varphi} \tag{4.10}$$

where b_{mult}^{v} is the velocity multiplicative bias, b_{mult}^{φ} is the steering wheel angle multiplicative bias and b_{add}^{φ} is the wheel angle additive bias. The authors employed a PSO-based optimization method using a fitness function that computes the difference between GPS and dead reckoning poses. Each particle contains the odometry bias parameters, $\boldsymbol{B} = (b_{mult}^{v}, b_{mult}^{\varphi}, b_{add}^{\varphi})$, and the best parameters are used to update all odometry measurements before the AV poses optimization.

We propose a new iterative method to calibrate the odometry bias. The odometry calibration is is executed directly in the hypergraph optimization.

An odometry bias calibration node, $\boldsymbol{N}^B_{j:k}$, contains the parameters of the Eqs. 4.9 and 4.10:

$$\boldsymbol{N}^B_{j:k} = (b^v_{mult}, b^\varphi_{mult}, b^\varphi_{add})_{j:k} \tag{4.11}$$

The range subscript $j{:}k$ indicates that a single bias calibration node $\boldsymbol{N}^B_{j:k}$ can be related to any subgroup of AV poses $\boldsymbol{x}_{j:k}$ containing all consecutive poses from j to k. Considering n as the total number of nodes in the hypergraph, a single bias calibration node $\boldsymbol{N}^B_{1:n}$ is connected to all poses, meaning that the system applies the same calibration parameters to all odometry measurements. Otherwise, a pair of nodes $\boldsymbol{N}^B_{1:n/2}$ and $\boldsymbol{N}^B_{n/2:n}$ can separate the odometry measurements in two different parts of equal sizes and find different biases along the same log or session. As each session generates its own hypergraph and each hypergraph have at least one bias calibration node, the system also handle the bias changes between sessions.

Each pair $(\boldsymbol{N}_i, \boldsymbol{N}_{i+1})$ of nodes generated by the sensor measurements is linked to a calibration node $\boldsymbol{N}^B_{j:k}$ by an odometry bias calibration edge. This special edge connects three nodes and by definition it is a hyperedge. In order to calibrate the odometry bias we use the following error:

$$\boldsymbol{e}_{i,j:k}\left(\boldsymbol{x}, \boldsymbol{B}, \boldsymbol{u}\right) = \left(\boldsymbol{x}_{i+1} \ominus \boldsymbol{x}_i\right) \ominus \overline{K}(\boldsymbol{B}_{j:k}, \boldsymbol{u}_i, \Delta t) \tag{4.12}$$

where $\overline{K}(.)$ is a function that takes the bias parameters $\boldsymbol{B}_{j:k}$ at $\boldsymbol{N}^B_{j:k}$ to update the command $\boldsymbol{u}_i$ through Eqs. 4.9 and 4.10 then calls $K(.)$ to estimate a new relative movement. Note that the range $j{:}k$ must include i and $i{+}1$. In section 4.4 we indicate how and when this error is evaluated. All errors computed by the odometry bias calibration edges define the total odometry bias calibration error:

$$\begin{aligned}\boldsymbol{J}^B(\boldsymbol{x}, \boldsymbol{B}, \boldsymbol{u}) = \\ \sum_{j:k} \sum_{i=j}^{k-1} \boldsymbol{e}_{i,j:k}\left(\boldsymbol{x}, \boldsymbol{B}, \boldsymbol{u}\right)^\intercal \hat{\boldsymbol{\Omega}}^B_i \boldsymbol{e}_{i,j:k}\left(\boldsymbol{x}, \boldsymbol{B}, \boldsymbol{u}\right)\end{aligned} \tag{4.13}$$

where $\hat{\Omega}^B_i$ is the odometry information matrix at node $\boldsymbol{N}_i$ projected through the non-linear function $\overline{K}(.)$ via the unscented transformation. In this case, the projection must be updated when the bias changes substantially.

The resulting hypergraph contains all pose nodes, the odometry bias calibration nodes and all required edges to impose the constraints defined in Eqs. 4.4, 4.6, 4.8 and 4.13. Fig. 11 illustrates the relationships between the elements in the hypergraph, including some pose nodes, some odometry edges, a GPS edge and a loop closure edge. Fig. 12

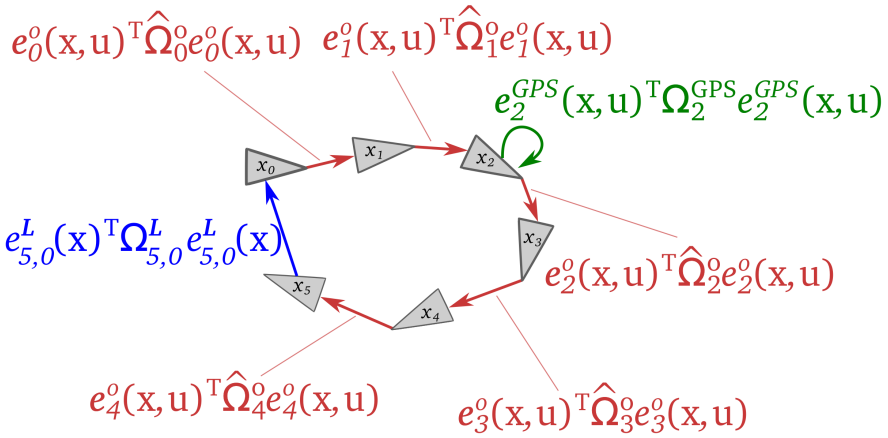


Figure 11 – Basic graph.
Graph containing edges from odometry, GPS and loop closure.

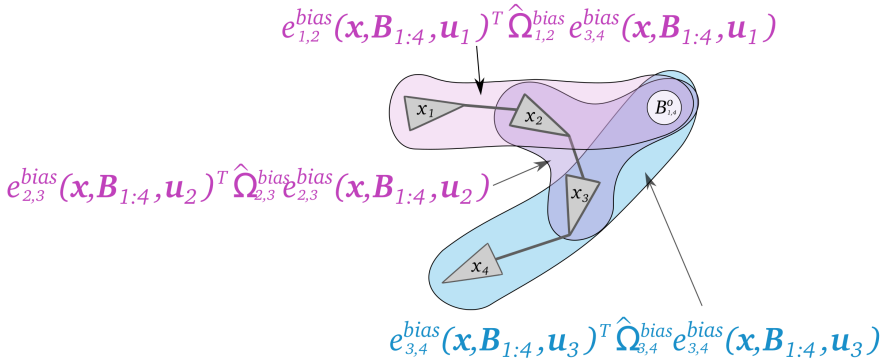


Figure 12 – The hypergraph.
Odometry bias calibration node and hyperedges.

displays an odometry bias calibration node and some calibration hyperedges represented by regions with different colors for better visualization.

4.3.5 Lidar Odometry Edges

Lidar odometry (LO) is usually treated as a registration problem between two point clouds. The registration problem consists of finding a linear transform that aligns two point clouds. Fig. 13 shows two clouds with different colors. Initially, the blue and green point clouds are in the sensor coordinate frame, then a registration algorithm is used to find the linear transform that align them. After the alignment, one can estimate the sensor movement using the transform between the two clouds. This works better when the environment contains only static objects since moving objects can deteriorate or even forbid a good estimate of the sensor movement. We propose a new method to estimate the LO in the presence of moving objects. The method estimates the movement of the LiDAR sensor and then it adds LO edges to the graph. A LO edge connects two pose nodes that

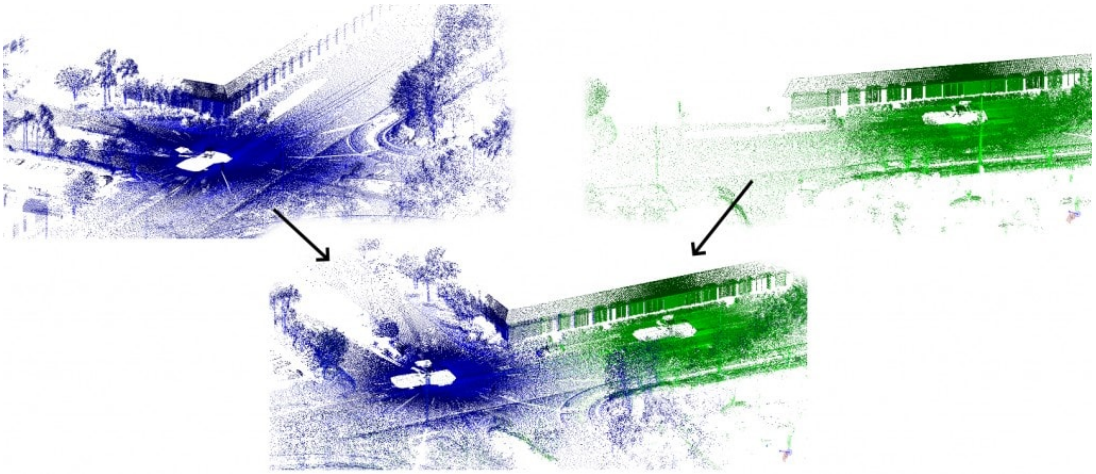


Figure 13 – Point Cloud Registration

were generated by LiDAR measurements.

The method starts by taking an initial point cloud $\boldsymbol{z}_i^{LiDAR}$ and then selecting the next point cloud $\boldsymbol{z}_j^{LiDAR}$ using the same approach described in [35]. As in [35], we consider the frequency of the LiDAR sensor and the vehicle velocity to compute the index of the next point cloud. Any point cloud between the selected ones is discarded. After loading the target point clouds, the method uses the simplified semantic segmentation described on [36] to remove most of the moving objects and increase the quality of the point cloud registration. After the segmentation, we estimate an initial guess for the movement of the LiDAR sensor by taking the odometry measurements between $\boldsymbol{z}_i^{LiDAR}$ and $\boldsymbol{z}_j^{LiDAR}$ and computing the vehicle movement. The two point clouds and the initial guess are used as inputs to the GICP algorithm. The output of the GICP is a linear transform that is used to build a new LO edge connecting the nodes related to $\boldsymbol{z}_i^{LiDAR}$ and $\boldsymbol{z}_j^{LiDAR}$. After adding the current LO edge, the method combines $\boldsymbol{z}_i^{LiDAR}$ and $\boldsymbol{z}_j^{LiDAR}$ into a single point cloud $\boldsymbol{z}_{i\cup j}^{LiDAR}$ and use the new cloud in the next iteration. The next iteration repeats the process by considering $\boldsymbol{z}_{i\cup j}^{LiDAR}$ as reference and selecting the next point cloud $\boldsymbol{z}_k^{LiDAR}$. Note that $\boldsymbol{z}_{i\cup j}^{LiDAR}$ and $\boldsymbol{z}_k^{LiDAR}$ will be combined into $\boldsymbol{z}_{i\cup j\cup k}^{LiDAR}$ later, therefore, the density of the combined point clouds must be reduced from time to time to avoid memory issues. For that, the method uses a voxel grid filter while also removing all points that are too far away from the center of the last point cloud.

This new LO method is able to estimate the vehicle poses with higher accuracy than that obtained solely by the calibrated odometry. The LO edges have the same error defined in 4.7 and they are all combined in the global LO error:

$$\boldsymbol{J}^{LO}(\boldsymbol{x}) = \sum_i (\boldsymbol{e}_{i,j}^{LO}(\boldsymbol{x}))^\intercal \boldsymbol{\Omega}_{i,j}^{LO} (\boldsymbol{e}_{i,j}^{LO}(\boldsymbol{x})) \tag{4.14}$$

where Ω_i^{LO} is the inverse of the GICP covariance matrix.

The moving objects in the environment may induce wrong results in a naive LO algorithm. For example, the AV may be stopped but the moving objects around the AV can be seen by the LiDAR sensor and their corresponding point clouds would indicate some unreal movement of the sensor. Therefore, a typical LO method would benefit from a 3D point cloud semantic segmentation that eventually can classify and remove moving objects from the point clouds. Section 4.3.5.1 presents a simplified semantic segmentation that was implemented in this project.

4.3.5.1 Simplified Semantic Segmentation

The proposed system implements part of the approach on virtual city reconstruction proposed by [36]. Their method is able to create 4D spatio-temporal models or large dynamic urban scenes. One of their objectives was to reconstruct urban environments including only the ground and tall structures. The authors proposed a point cloud segmentation method to remove undesired objects and improve the point cloud registration. Their segmentation method was replicated in this project to enhance our custom LiDAR odometry algorithm. The segmentation assign each point to one of the following classes: (i) *clutter*, (ii) *ground*, (iii) *tall structure objects* and (iv) *short street objects*. They also handle vegetation in a separate method that was not replicated in this project.

The *clutter* class identifies isolated points that are too distant from their nearest neighbor. The *ground* class aggregates points that belong to the road, sidewalks or traffic islands. The *tall structure objects* class aims to separate objects that are usually static as buildings, walls, roofs, lamps posts and traffic lights. The *short street objects* class is used to identify most moving objects as vehicles and pedestrians.

A grid based approach is employed to achieve the point cloud segmentation. For each point cloud $\boldsymbol{\mathcal{P}}$, the method fits a 2D grid $\boldsymbol{\mathcal{G}}$ onto the $z = 0$ plane. The grid size is set to a width between $50cm$ and $80cm$. Smaller cell sizes produce many empty space or cells containing few points and no statistical relevance. Larger cell sizes can result on several falsely classified points, given that different objects can be projected to the same cell. The dimension of the grid is obtained from the point cloud $\boldsymbol{\mathcal{P}}$.

Each point $p \in \boldsymbol{\mathcal{P}}$ is projected to a given cell $c_p \in \boldsymbol{\mathcal{G}}$. Let $\mathcal{P}_c \in \boldsymbol{\mathcal{P}} : \boldsymbol{c = c_p}$ denote the point set projected to cell c. The functions $h_{max}(c)$, $h_{min}(c)$ and $h_{avg}(c)$ return respectively the maximum, minimum and average of the elevation values within $\mathcal{P}_c$.

After projecting all points to the grid, the method tries to find the points that belongs to the *clutter* class. For each cell c, the method calculates the local density by summing the total number of points projected to c and its 8 surrounding cells. If the quantity is below a threshold value τ, then the points are classified as *clutter* and they are

discarded. The following expression indicates how to classify points in the *clutter* class:

$$\tau > p_{count}(c) + \sum_{r \in N_c^n} p_{count}(r) \rightarrow p_{clutter} \tag{4.15}$$

where N_c^n is the n neighbor cells, in this project we used $n = 8$. The function $p_{count}(\cdot)$ returns how many points were projected to a given cell.

By visiting the cells around c, the method handles larger objects that may have many points on several cells but just few points projected to c.

The next step tries to classify all points that belong to the *ground* class. First, the method finds the maximum difference between the lowest and the highest point from a point set $\mathcal{P}_c$:

$$\Delta h = h_{max}(c) - h_{min}(c) \tag{4.16}$$

If the difference Δh is lesser than a threshold value τ_{gr}, then it may indicate that the points projected to c belongs to the *ground* class. The threshold value τ_{gr} means the max allowed Δh to consider a cell as ground candidate. In our project, we used $\tau_{gr} = 0.25m$. However, this condition is not sufficient given that other structures may have similar Δh (e.g. flat car roof and engine hood). This initial test is used to classify the ground candidates:

$$\begin{cases} \mathbf{1}_G(c) = 1 & if \Delta h < \tau_{gr} \\ \mathbf{1}_G(c) = 0 & otherwise \end{cases} \tag{4.17}$$

where $\mathbf{1}_G(c) = 1$ is a ground candidate and $\mathbf{1}_G(c) = 0$ is an undefined region.

The final classification is done by removing the outliers using a spatial filtering. Their algorithm assumes that the outliers, such as a flat car roof, will be projected to a few cells and, on the other hand, the ground will be projected to many cells on $\boldsymbol{\mathcal{G}}$.

Considering N_c^v as the $v \times v$ neighborhood of c, and $\gamma_G(c) = \sum_{r \in N_c^v} \mathbf{1}_G(c)$, they define a terrain model of the scene:

$$h_{gr}(c) = \begin{cases} \frac{1}{\gamma_G(c)} \cdot \sum_{r \in N_c^v} h_{avg}(c) \cdot \mathbf{1}_G(c), & \text{if } \gamma_G(c) > 0 \\ undefined, & otherwise \end{cases} \tag{4.18}$$

The authors used a neighborhood size v equals to 17. A cell c is considered as ground cell if it is a ground candidate and the difference between $h_{avg}(c)$ and $h_{gr(}c)$ is

lesser $0.2m$:

$$\mathbf{1}_G(c) = 1 \wedge h_{avg}(c) - h_{gr}(c) < 0.2m \rightarrow c_{ground} \tag{4.19}$$

The *tall structure objects* class requires a simpler analysis:

$$1.4m < h_{max}(c) \vee h_{max}(c) - h_{min}(c) > 3.10m \rightarrow c_{tall} \tag{4.20}$$

The expression means that a cell is *tall structure object* if the maximal observed elevation is larger than $1.4m$ or if the difference between the maximal and minimal elevations on c is larger than $3.10m$. The second criteria is useful when there are objects standing on a lower point when compared to the ground.

After classifying the points on *clutter*, *ground* or *tall structure objects*, the remaining cells are considered as *short street objects* and all points projected to these cells are removed from the point cloud. Although this method does not classify small objects as static and moving objects, it is able to remove most moving objects from the scene while keeping the ground and the tall objects.

4.4 Hypergraph Optimization

The hypergraph optimization is an iterative process which is repeated for a predefined number of iterations. The optimization is separated in two main steps. In the first step, the optimization considers the sum of the errors related only to the AV poses. In other words, only the nodes regarding the AV poses are updated while all the nodes related to the odometry bias remains fixed. This step maximize the likelihood of the AV poses by combining Eqs. 4.4, 4.6, 4.8 and 4.14 and excluding the error defined in Eq. 4.13:

$$\boldsymbol{J}_1(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{J}^o(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{J}^{GPS}(\boldsymbol{x}) + \boldsymbol{J}^L(\boldsymbol{x}) + \boldsymbol{J}^{LO}(\boldsymbol{x}) \tag{4.21}$$

The inter-session loops are included in Eq. 4.21, however the system also optimizes each subgraph with their related intra-session loops in a separated manner. After the optimization of the subgraphs, the system unifies all subgraphs into a single global graph using the inter-session loops and proceeds to the global optimization.

When the first step reaches a defined number of iterations, the optimization process swaps the error functions. The second step keeps all nodes related to the AV pose fixed while only the odometry bias calibration nodes are updated. All errors computed by Eq. 4.21 are excluded and the global error now considers only Eq. 4.13:

$$\boldsymbol{J}_2 = \boldsymbol{J}^B(\boldsymbol{x}, \boldsymbol{B}_{j:k}, \boldsymbol{u}) \tag{4.22}$$

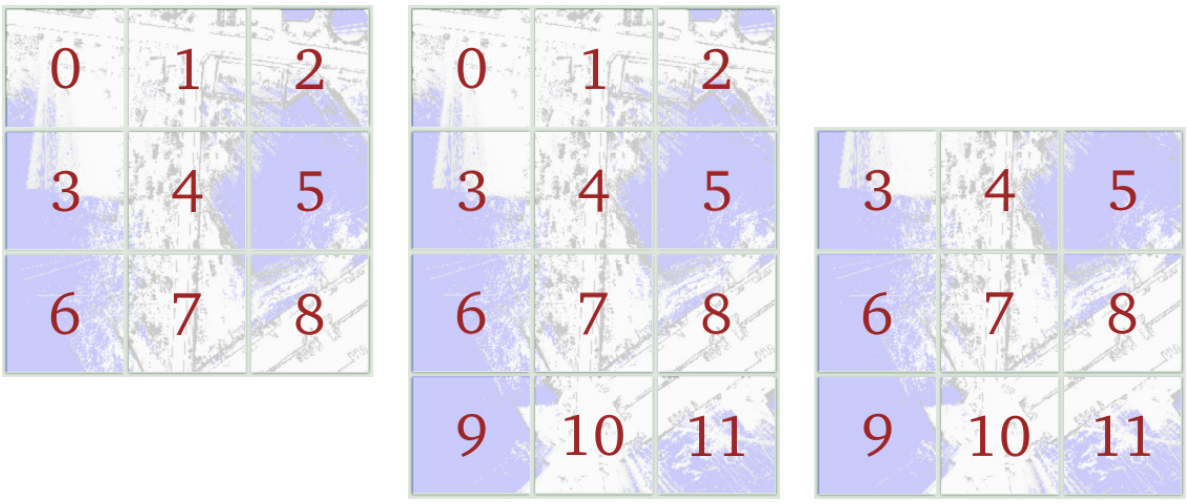
When the second step reaches a defined number of iterations, the new odometry biases can be used to update all corresponding commands $\boldsymbol{u}$ using Eqs. 4.9 and 4.10. All the odometry edges related to Eq. 4.7 can be updated with the new commands and the optimization process can restart from the first step.

After repeating the two steps of optimization for a number of iterations and assuming convergence, the optimization process returns for each log the AVs poses and odometry calibration biases. The mapper module receives the AVs poses to create a new map or update an existing one.

4.5 Large-scale Environment Mapping

The mapper system must handle high memory usage by virtue of the small grid sizes, the large-scale environments and also the simultaneous output of grid maps of many types. Thus, we split the grid maps in blocks of 75 x 75 meters and for each grid map we keep only nine blocks in memory. Fig. 14 shows the sub-map approach. We use a sub-map manager to always maintain the AV in the central block. When the AV crosses the central block boundaries the sub-map manager loads the proper neighbor blocks from the hard disk (HD) and saves the unused blocks to HD and frees memory. This sub-map technique guarantees a constant map of at least 225 x 225 meters and smooth map transitions.

The occupancy grid map (OGM) estimates the posterior probability of the map



(a) (b) (c)

Figure 14 – Sub-map technique.
(a) 225 x 225 meters map in main memory. The robot is in the central block 4. (b) The robot moves to block 7 and some new block are loaded from disk. (c) At this moment, the upper blocks are saved in the disk and removed from the main memory

given all AV poses and sensor measurements:

$$p\left(\boldsymbol{m}|\boldsymbol{x},\boldsymbol{z}\right) \tag{4.23}$$

Let $\boldsymbol{m}_c$ be the grid cell with index c and let $p\left(\boldsymbol{m}_c\right)$ be the probability that a grid cell is occupied. A threshold value is used to to consider if a cell is an obstacle, a free space or unknown. If $p\left(\boldsymbol{m}_c\right)$ is greater than a desired threshold (e.g., 0.5) then we assume $\boldsymbol{m}_c$ is an obstacle. If $p\left(\boldsymbol{m}_c\right)$ is less than the threshold value then $\boldsymbol{m}_c$ is considered a free space. Finally, we assume that the state of a grid cell $\boldsymbol{m}_c$ is unknown if it was never observed.

As it is defined in Eq. 4.23, estimating the posterior probability of a single map becomes intractable when map size and resolution increases. Our OGM contains 140625 grid cells in each block, therefore we would need to compute the posterior probability of 2^{140625} grid maps for each block. We follow the standard of assuming that the cells probabilities are independent from each other in order to estimate the map in a collection of separated problems [37]. The map posterior is then approximated by the product of its marginals:

$$p\left(\boldsymbol{m}|\boldsymbol{x},\boldsymbol{z}\right) = \prod_c p\left(\boldsymbol{m}_c|\boldsymbol{x},\boldsymbol{z}\right) \tag{4.24}$$

The map is represented in the log-odds format in order to avoid numerical instabilities due to near to zero or near to one probability [30]. The log-odds of a given given cell $\boldsymbol{m}_c$ is defined as:

$$l_c = \log \frac{p\left(\boldsymbol{m}_c|\boldsymbol{x},\boldsymbol{z}\right)}{1 - p\left(\boldsymbol{m}_c|\boldsymbol{x},\boldsymbol{z}\right)} \tag{4.25}$$

The probability of $\boldsymbol{m}_c$ being occupied can be recovered through the inverse function:

$$p\left(\boldsymbol{m}_c|\boldsymbol{x},\boldsymbol{z}\right) = 1 - \frac{1}{1 + \exp l_{i,c}} \tag{4.26}$$

The algorithm computes the occupancy grid map in an iterative way. Initially, all grid cells are set to an invalid state (e.g. -1.0) and, at each time step, if a cell $\boldsymbol{m}_c$ is in the LiDAR perceptual field then it's current log-odds is updated:

$$l_{i,c} = l_{i-1,c} + ISM\left(\boldsymbol{m}_c, \boldsymbol{x}_i, \boldsymbol{z}_i^L\right) - l_0 \tag{4.27}$$

where $l_{i-1,c}$ is the previous log-odds of the cell $\boldsymbol{m}_c$, l_0 is the map prior probability and $ISM(.)$ is the inverse sensor model function which returns the probability of the measurement $\boldsymbol{z}_i^L$ in the log-odds format, given the current AV pose $\boldsymbol{x}_i$ and cell $\boldsymbol{m}_c$. Usually, $ISM(.)$ is an expensive function, thus many different ad-hoc or heuristic methods are

applied instead [19]. We describe next our custom ad-hoc method to compute the obstacle evidence (OE) using LiDAR rays.

4.6 Obstacle Detection

The Mapper module uses LiDAR sensors to compute the probabilities in the OGM. Our main LiDAR is a Velodyne HDL-32E placed above the AV. This sensor contains 32 vertical lasers across a 40º field of view. The Velodyne rotates around itself and it provides point clouds with with 360º horizontal field of view. Considering the point clouds in spherical coordinates, $b_n^{i,j}$ represents the i^{th} beam range of the j^{th} vertical set of readings in the n^{th} point cloud. For each pair of beams $b_n^{i,j}$ and $b_n^{i+1,j}$, the Mapper estimates the obstacle evidence (OE) of the ray $b_n^{i+1,j}$ using the following definition:

$$OE(b_n^{i,j}) = \frac{ED\left(b_n^{i,j}, \hat{b}_n^{i+1,j}\right) - MD\left(b_n^{i,j}, b_n^{i+1,j}\right)}{ED\left(b_n^{i,j}, \hat{b}_n^{i+1,j}\right)} \tag{4.28}$$

where $\hat{b}_n^{i+1,j}$ is an estimation of beam of $b_n^{i+1,j}$ in an obstacle free environment, $ED(.)$ is the expected absolute difference between the projections of $b_n^{i,j}$ and $b_n^{i+1,j}$ on the ground and $MD(.)$ is the absolute difference of the actual sensor measurements $b_n^{i,j}$ and $b_n^{i+1,j}$ projected on the ground. Fig. 15 shows the projections and the relationships between $ED(.)$ and $MD(.)$.

The $MD(.)$ value is obtained from:

$$MD\left(b_n^{i,j}, b_n^{i+1,j}\right) = \left|proj\left(b_n^{i+1,j}\right) - proj\left(b_n^{i,j}\right)\right| \tag{4.29}$$

where $proj(.)$ is a function that returns the length of the projection on the ground. The $ED(.)$ function is computed through LiDAR sensor height from the ground h, the measured length r of $b_n^{i,j}$ and the vertical angle ω between $b_n^{i,j}$ and $b_n^{i+1,j}$. First, the method recovers the vertical angle of $b_n^{i,j}$:

$$\alpha = \arccos\left(\frac{h}{r}\right) \tag{4.30}$$

Then, the projection of $b_n^{i,j}$ on the ground is computed by:

$$proj\left(b_n^{i,j}\right) = r \times \sin\left(\alpha\right) \tag{4.31}$$

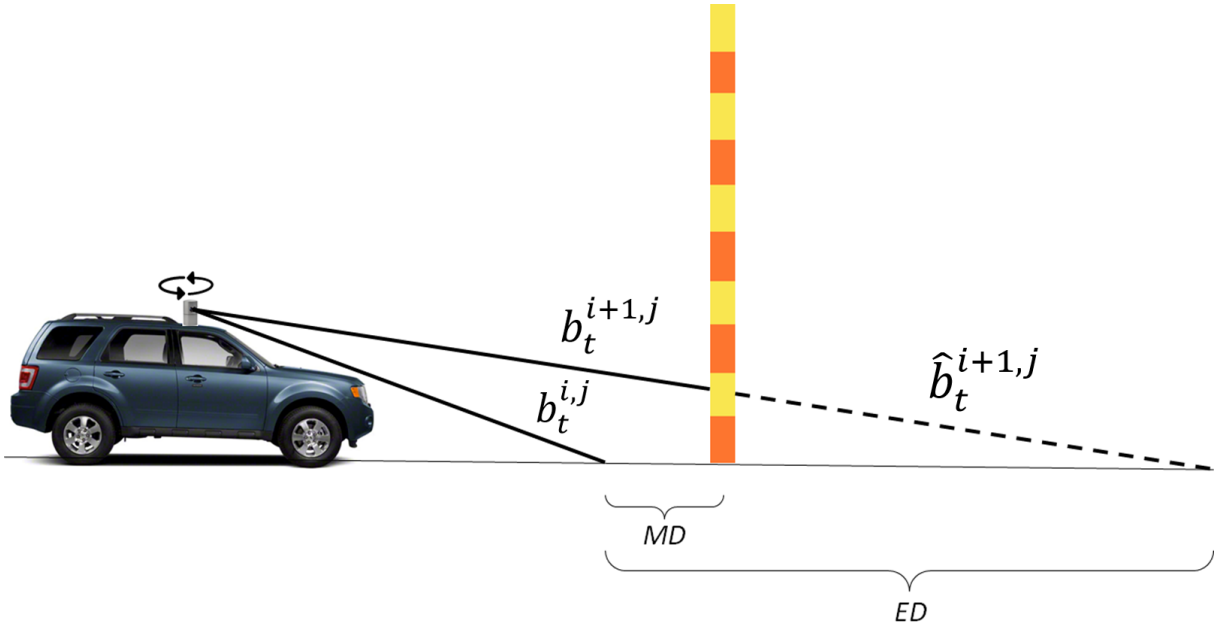


Figure 15 – Obstacle evidence.
Illustration of the obstacle evidence approach. Two LiDAR rays are projected in the ground and the difference between the projections length is the actual measured distance MD(.). The expected distance ED(.) is estimated by subtracting the length of $b_n^{i,j}$ in the ground from the expected length of $b_n^{i+1,j}$ in the ground.

Then, we estimate the projection of $\hat{b}_n^{i+1,j}$ on the ground in an obstacle free environment:

$$proj\left(\hat{b}_n^{i+1,j}\right) = \frac{\sin\left(\alpha+\omega\right)}{\sin\left(\frac{\pi}{2}+\alpha+\omega\right)} \tag{4.32}$$

Finally, we obtain the $ED(.)$ value using Eqs. 4.31 and 4.32:

$$ED\left(b_n^{i,j}, \hat{b}_n^{i+1,j}\right) = \left|proj\left(\hat{b}_n^{i+1,j}\right) - proj\left(b_n^{i,j}\right)\right| \tag{4.33}$$

As in [3] we compute the obstacle evidence $OE(.)$ defined in Eq. 4.28 using Eqs. 4.30 and 4.33 in order to obtain the probability:

$$PO = \frac{1}{\sqrt{2\pi\sigma_{OD}^2}} \exp\left(\frac{-\left(1-OE\left(b_n^{i+1,j}\right)\right)^2}{2\sigma_{OD}^2}\right) \tag{4.34}$$

where σ_{OD} is the obstacle detection standard deviation found empirically in [3]. The last step converts the probability value computed by Eq. 4.34 to the log-odds format and updates the grid maps cells in the LiDAR sensor field of view.

4.7 Large-scale Map Merging

The Mapper module merges the grids maps in an incremental approach. A first map is created using any of the resulting output files from the hypergraph optimization. A good practice is to start the mapping process using the session recorded in the best conditions (i.e few moving objects). The sessions can be sorted by the moving objects quantity and organized in a queue where the session with less moving objects are consumed first.

While the mapping process evolves, the system conveniently saves how many times each grid map cell $\boldsymbol{m}_c$ is reached by the LiDAR rays in the current session. After the first map is complete, the system changes the rule defined in Eq. 4.27 and then only the cells with the hit counter below a threshold value K are updated:

$$l_{i,c} = \begin{cases} l_{t-1,i} + ISM\left(\boldsymbol{m}_c, \boldsymbol{x}_i, \boldsymbol{z}_i^L\right) - l_0, & hits\left(\boldsymbol{m}_c\right) < K \\ l_{t-1,i}, & otherwise \end{cases} \tag{4.35}$$

where K is any integer value greater than zero. We assume that the LiDAR sensor is reliable and after enough hits the system can safely freeze the resulting log-odds. For example, the cells containing static obstacles will quickly converge to high probabilities in the occupancy grid maps and so we can trust in those probabilities after K hits. Therefore, we don't update the cells which were very observed in the previous sessions. This procedure can unintentionally keep the low dynamics fixed in a previous map, but the system can easily return to the original update rule defined in Eq. 4.27 if the difference in time between the sessions are greater than a desired threshold $InterSessionTimeDiff$. Assuming that the main changes in the environment infrastructure takes a time greater than $InterSessionTimeDiff$, then the system can update the current map while discarding most of the undesired changes in the incoming maps.

5 Experimental Methodology

5.1 IARA

The proposed mapping system was designed to the Intelligent Autonomous Robotics Automobile (IARA, 2). IARA is an AV developed by the computational intelligence research group of the Laboratório de Computação de Alto Desempenho - LCAD (High Performance Computing Laboratory, `http://www.lcad.inf.ufes.br`) at the Universidade Federal do Espírito Santo - UFES (Federal University of Espírito Santo, `http://www.ufes.br`), in Vitória, Brazil. IARA is based on a Ford Escape Hybrid, which was modified to allow electronic control of steering, throttle, brakes, gears and several signalization items; and to provide the car odometry for the IARA's autonomy system, and power supply for computers and sensors. Its main computer is a Dell Precision R5500 with two Xeon X5690 six-core 3.4 GHz processors and one NVIDIA TITAN Xp. Its sensors include one Velodyne HDL 32-E LIDAR, one Trimble RTK GPS, one Xsens MTi IMU and one Bumblebee XB3 stereo camera. The IARA's autonomy system follows the typical architecture of self-driving cars [2]. It is based on the Carnegie Mellon Robot Navigation Toolkit (CARMEN [38]), which is a modular open source software collection for mobile robot control. We have significantly extended and currently maintain a version of CARMEN, available at `https://github.com/LCAD-UFES/carmen_lcad`. For details on the IARA's autonomy system, readers are referred to Badue et al. [2].

5.2 Sensor Data Logs

To evaluate the performance of the proposed mapping system, we built several sensor data logs. We collected data from all IARA's sensors while IARA was being conducted by a human driver along the beltway of the UFES main campus, with about 3.7 km of extension, and within parking lots in the vicinity of the UFES campus beltway. The sensor data acquired on travels in the UFES campus were saved in several logs, which were used to assess the map merging capabilities of the proposed mapping system. We also captured IARA's sensor data while IARA was driven along a route from the UFES campus in the city of Vitória to the city of Guarapari, with about 72 km of extension. The sensor data acquired on the travel to Guarapari were saved in a single log, which was used to analyse the proposed mapping system in large-scale environments.

The logs were collected in different traffic conditions and contains a significant diversity of road infrastructure (e.g., bridges, buildings, highways, crossings, etc). They were also captured at different times - in the UFES campus, over a month, and on the

Figure 16 – UFES main campus beltway.
Satellite view of the main campus (in red).

Figure 17 – Parking lot 1 (in yellow).

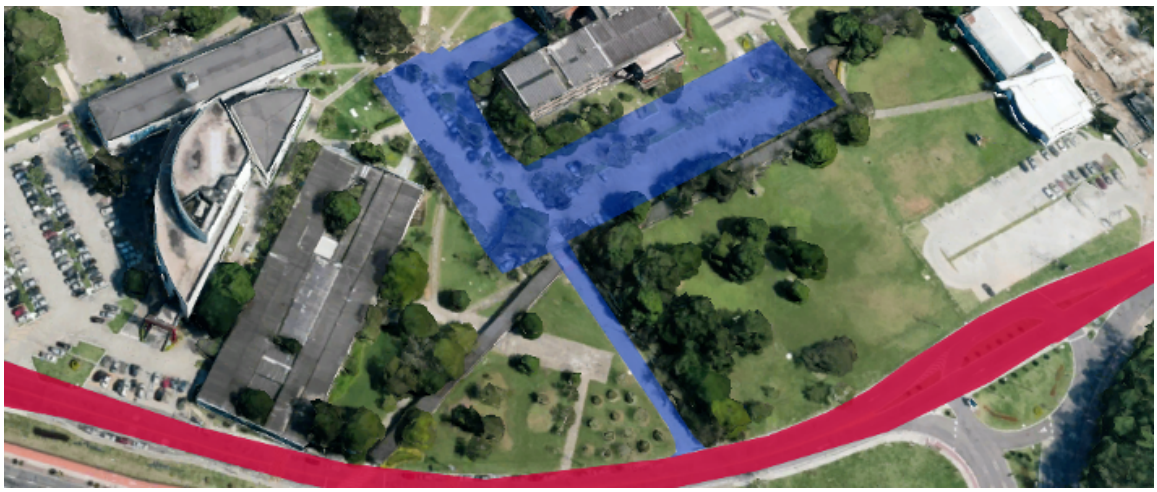


Figure 18 – Parking lot 2 (in blue).

travel to Guarapari, two years before.

Figures. 16, 17, 18, 19 and 20 show satellite views of the environments from

Figure 19 – Parking lot 3 (in cyan).

Figure 20 – Route from UFES to Guarapari.

which the logs were collected. Fig. 16 shows the UFES campus beltway (in red), which is surrounded by trees in all its extension. The logs UB1 and UB2 were recorded while the driver conducted IARA along the beltway; the log UB1 was acquired at night, in the clockwise direction and without moving obstacles, while the log UB2 in the morning, in counterclockwise direction and with moving objects (pedestrians, cars, bicycles, etc.) and many vehicles parked. It is important to note that the logs UB1 and UB2 present inter-log loop closures along the entire path.

Fig. 17 shows the parking lot 1 (in yellow), Fig. 18 the parking lot 2 (in dark blue) and Fig. 19 the parking lot 3 (in cyan), which are also surrounded by trees and impose significantly higher quantity of errors in GPS measurements when compared to the beltway. The logs PL1, PL2 and PL3 were recorded while the driver conducted IARA within the parking lots 1, 2 and 3, respectively.

Finally, Fig. 20 shows the route from the UFES campus in the city of Vitória to the city of Guarapari (in light blue), which contains many different environments as the AV crosses different cities, highways, rural and urban streets, and a bridge of about 3.3 km of extension. The log TRVL was recorded while the driver conducted IARA along the

route from Vitória to Guarapari.

We use the proposed mapping system to create grid maps and merge them using the logs collected along the courses in the UFES campus and the route from Vitória to Guarapari. Although the logs were captured in a wide time range, the proposed mapping system was able to create an accurate global map using them, which allowed IARA to navigate and self-locate in the mapped environments during later travels.

5.2.1 Parameters

Table 1 shows the main parameters of the proposed mapping system used in our experiments.

5.3 Metrics

We used the mean absolute error (MAE) to compare two paths and verify how close they are from each other. We usually consider the GPS as the ground truth path to be compared with paths computed from any other sensor or method.

We illustrate the same obstacles viewed in different visits to the same region with different colors to qualitatively evaluate the drifts and misalignment in the resulting grid maps. This coloring scheme also facilitates the verification of the contribution of loop closures and allows the visualization of the impact of the new update rule of a grid map cell defined in Eq. 4.35 to the quality of the resulting grid maps.

Table 1 – Main parameters of the proposed mapping system used in our experiments.

Name	**Value**	**Description**
$MinSpeed$	$0.01ms$	Minimum AV speed used in the speed filtering
$NeighborDistance$	$1.2m$	Distance threshold in the GPS clustering. Higher distances means disconectivity
$NeighborQuantity$	40	Quantity threshold in the GPS clustering. Clusters with fewer elements are removed
$LoopDistance$	$5.0m$	Maximum required distance between point clouds for loop closure candidate
$InterLoopTime$	$60sec$	Minimum required time between point clouds for inter-log loop closure candidate
K	64	Cell hits threshold for 4.35
$InterSessionTimeDiff$	$1month$	Time distance between sessions for low dynamics consideration
σ_{OD}	0.8	Obstacle detection standard deviation

5.4 Experiments

We carried out a series of experiments to evaluate the odometry calibration, the map building and the map merging proposed methods. Each experimental evaluation is detailed below.

5.4.1 Odometry Bias Calibration

In this experiment, we compare the PSO-based optimization [3] and the proposed hypergraph optimization methods for calibrating odometry bias. We use the MAE metric to indirectly compare the paths obtained by both odometry calibration methods against the GPS measurements. A better odometry calibration should produce lower MAE values as well the resulting path should better approximate the GPS measurements.

5.4.2 Map Building

In this experiment, for each log, we compare the estimated AV poses with the GPS measurements using the MAE metric. Although the GPS sensor does not provide an ideal ground truth, the GPS error is bounded and our current model uses RTK (Real Time Kinematics) corrections, so we can use it to verify if the estimated AV poses stay inside the GPS error. The quality of the grid maps is also visually verified using the coloring scheme.

The experiment also shows an ablation to demonstrate the impact of disabling the intra-session loop closures in the quality of the resulting grid maps.

5.4.3 Map Merging

In this experiment, we constructed a single map using the logs UB1, UB2, PLT1, PLT2 and PLT3, to verify the merging capabilities of the proposed mapping system.

The experiment also shows an ablation to demonstrate the effects of disabling either the inter-session loop closures or the new update rule of a grid map cell defined in Eq. 4.35 in the quality of the resulting grid map.

6 Results and Discussions

The results of each experiment are discussed in details in the following sections.

6.1 Odometry Bias Calibration

Table 2 describes the results of the odometry bias calibration using the PSO-based optimization [3] and the proposed hypergraph optimization methods.

Usually, both methods find relatively close values with an exception of the b^{φ}_{mult} in the TRVL dataset. The TRVL experiment exposes the impact of the sensor synchronization in the PSO as defined in the LEMS system. The LEMS approach finds the best odometry bias and also computes the dead reckoning considering only the odometer readings that are close in time to the LiDAR point clouds. This synchronization discards any additional odometry data between the point clouds that would be useful for computing the statistics. Differently, the new hypergraph approach obtains a path which is closer to the GPS when all relevant odometer readings are used.

Although the calibration values are relatively close in the remaining datasets, the corresponding paths are significantly different. This difference indicates how the dead reckoning is sensitive to the odometry bias. In all cases the hypergraph based approach have found better parameters since the recovered paths are closer to the GPS as indicated by the lower mean absolute errors (MAE).

Fig. 21 compares the GPS, the raw odometry and the odometry bias calibration in each dataset. The odometry calibration did not provide a substantial improvement in the parking lots (PL1, PL2 and PL3) but the paths recovered with the values from the hypergraph optimization are closer to the GPS than both raw odometry and PSO in all experiments.

We may note that the calibration methods use a linear model that do not consider the time variable. Therefore, it is not possible to handle any changes in the odometry bias in the TRVL dataset using a single node in the hypergraph.

6.2 Large-Scale Environmental Mapping

Table 3 shows the mean absolute errors (MAE) between the GPS and the estimated AV poses in each dataset. The lower MAE values indicate that the estimated AV poses stay close to the GPS. We have not found any AV poses outside the GPS limits in any experiment.

Table 2 – Odometry bias calibration using PSO and hypergraph optimization

LOG	Method	b^{v}_{mult}	b^{φ}_{mult}	b^{φ}_{add}	**MAE**
UB1	**PSO**	1.03011	0.97289	-1.412×10^{-3}	113.30952
	Graph	0.99663	1.02816	-1.530×10^{-3}	**38.00164**
UB2	**PSO**	0.99286	0.92143	0.499×10^{-3}	198.21247
	Graph	0.99647	1.00497	0.591×10^{-3}	**49.84211**
PL1	**PSO**	0.99091	0.98748	0.447×10^{-3}	5.61165
	Graph	0.99990	0.99752	0.747×10^{-3}	**2.35632**
PL2	**PSO**	1.00082	0.98382	0.354×10^{-3}	3.47713
	Graph	0.99675	0.99318	0.285×10^{-3}	**1.76424**
PL3	**PSO**	0.98799	0.98748	0.542×10^{-3}	2.08321
	Graph	0.99350	1.00022	0.562×10^{-3}	**1.77525**
TRVL	**PSO**	0.97999	**0.55000**	-0.304×10^{-2}	2.708×10^{4}
	Graph	1.00059	**0.98049**	-0.305×10^{-3}	$\mathbf{3.809\times10^{3}}$

Table 3 – Mean absolute error between GPS and AV poses.

LOG	**MAE**
UB1	0.250718
UB2	0.349465
PL1	0.326562
PL2	0.177750
PL3	0.318511
TRVL	0.123439

Fig. 22(a) and Fig. 22(b) exemplify the AV poses obtained by the hypergraph optimization in the UB1 log. The solid red line represents the GPS trajectory, the blue dashed line represents the final AV poses and the black solid line represents the GPS error limits. The results in the remaining datasets are very similar.

Fig. 23, Fig. 24, Fig. 25, Fig. 26, Fig. 26 and Fig. 28 present the occupancy grid maps of each dataset. Since the size of the map generated by the TRVL dataset does not allow a proper exhibition in small images, we decided to illustrate in Fig. 28 only a zoomed region in the beginning of the travel.

Given that our new system exploits all relevant sensor data, additional filtering strategies and LiDAR odometry, it was capable of handling the TRVL dataset and providing the required grid maps to the Guarapari project.

Fig. 22(c) illustrates the effects of disabling the intra-log loop closures in UB1 dataset. The estimated AV poses contains undesired mismatches in the loop closure region as these errors generate drifts in the grid maps. Fig. 29(a) illustrates the related drift and misalignment errors in occupancy grid map of UB1 and Fig. 29(b) displays the final grid map after activating the loop closures. The red color in the map indicates the first time the AV has detected an obstacle and the green color indicates the detection of the same

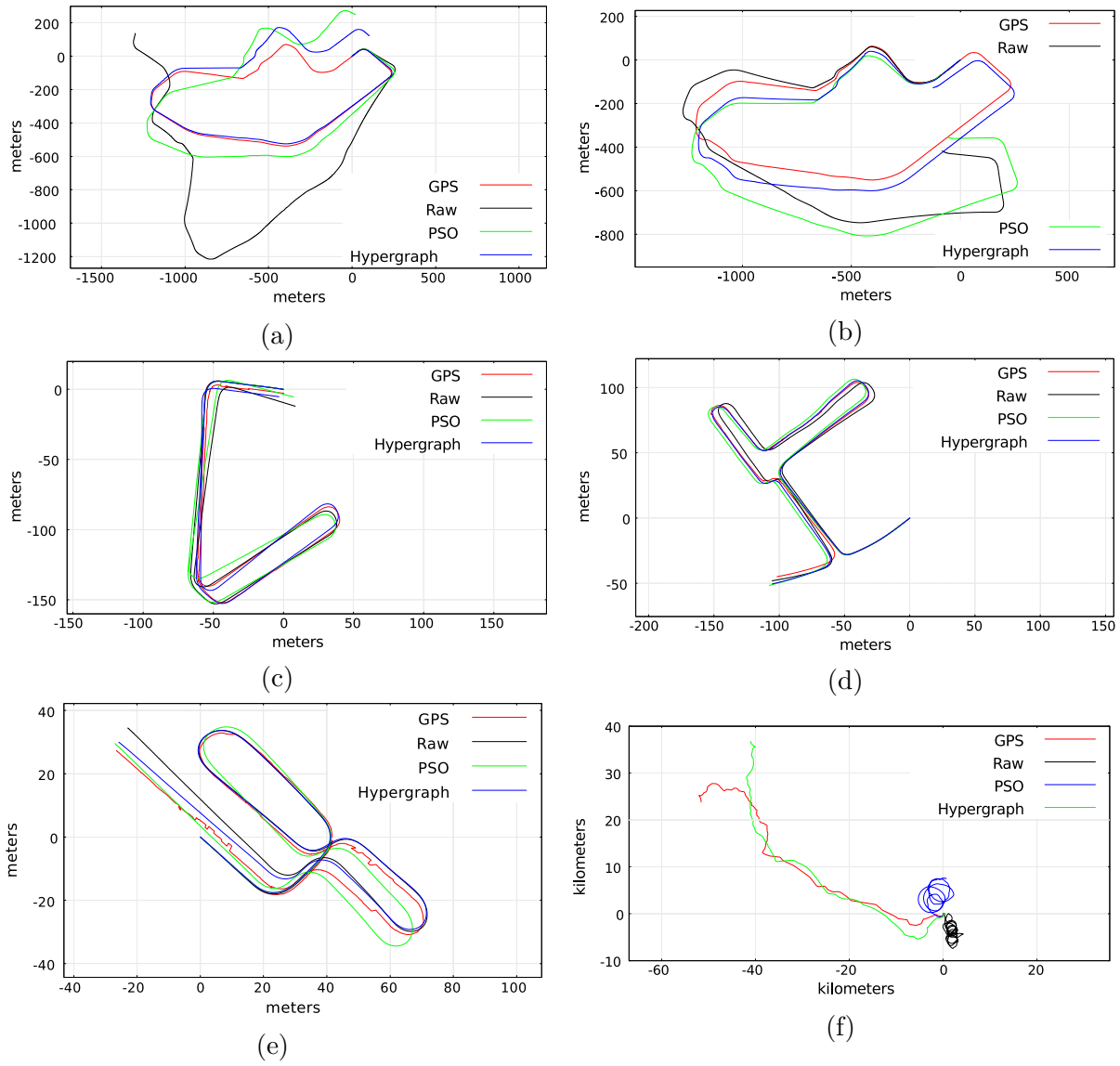


Figure 21 – Odometry calibration.
GPS, raw odometry and odometry calibration using PSO and the hypergraph optimization. (a) UB1. (b) UB2. (c) PL1. (d) PL2. (e) PL3. (f) TRVL.

obstacle later in the same log. The misalignment almost disappears after activating the intra-loop closures, this results demonstrate that the loop closures was crucial in the UB1 dataset. Fig. 29 also compares same effects in the remaining datasets. The TRVL dataset does not contain any loop closure since it was a direct path from the university to another city without revisited places.

6.3 Map Merging

We first illustrate the merging of the maps from Fig. 23 (UB1) and Fig. 24 (UB2) without the external loops while also using 4.35. Fig. 30(a) illustrates the lack of a precise common reference frame while merging the two maps. The red pixels are the obstacles

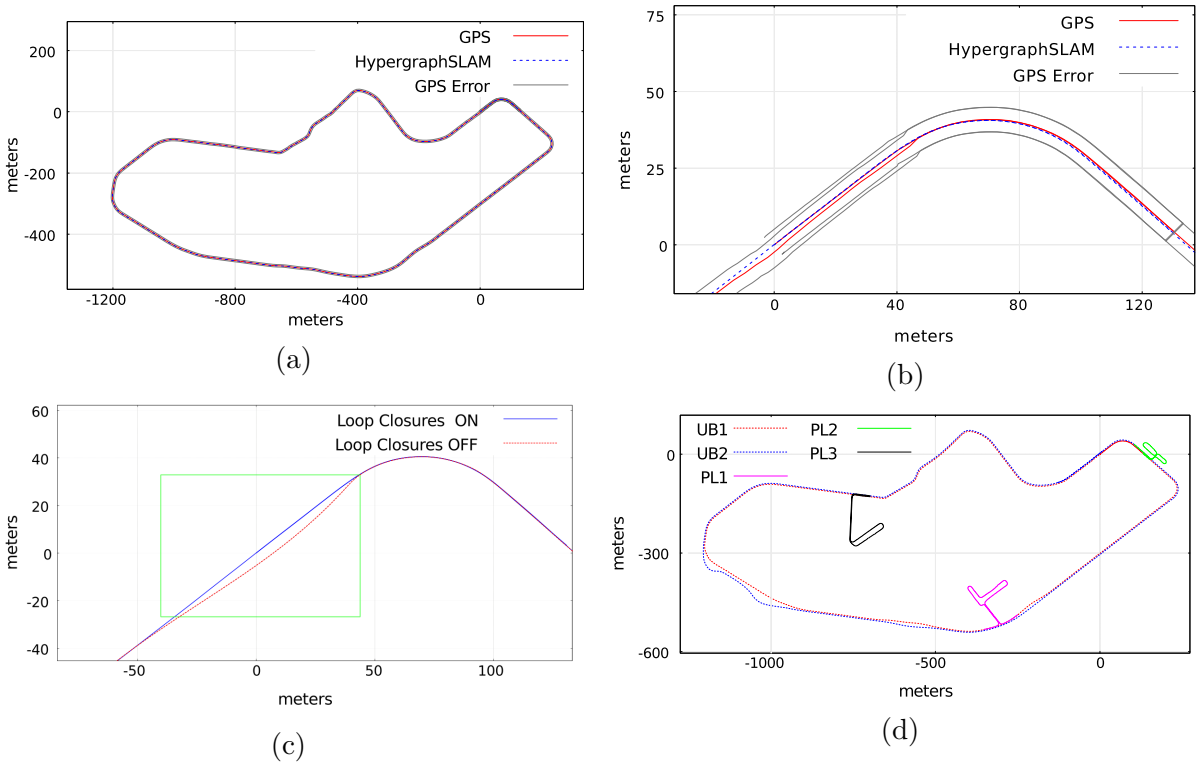


Figure 22 – AV poses optimization.
AV poses optimization. (a) UB1 - GPS and AV poses. (b) Loop closure region. (c) Toggling the intra-log loop closures (d) Final poses using all datasets.

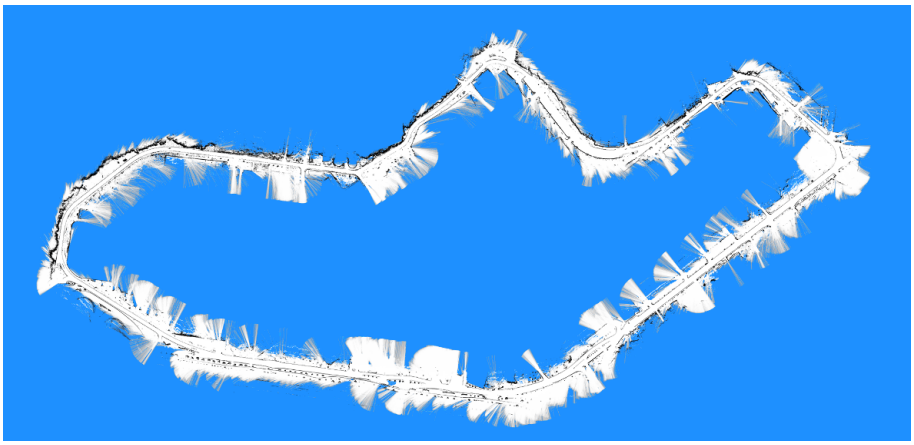


Figure 23 – Occupancy Grid Map - UB1

found in the UB1 log and the green pixels are the obstacles in UB2. We can see the drift between the logs through the curbs and buildings misalignment. The GPS measurements already moved the grid maps to the same region but the accuracy of our GPS sensor is not enough to provide the proper alignment. Similar errors occur when we merge the remaining parking lots into the university beltway without the loop closures. Fig. 30(b) illustrates the complete removal of the misalignment errors after activating the inter-log loop closures between UB1 and UB2. Fig. 30(c) presents the final merged map including all parking lots. Fig. 22(d) plots in the same reference frame all the optimized AV poses

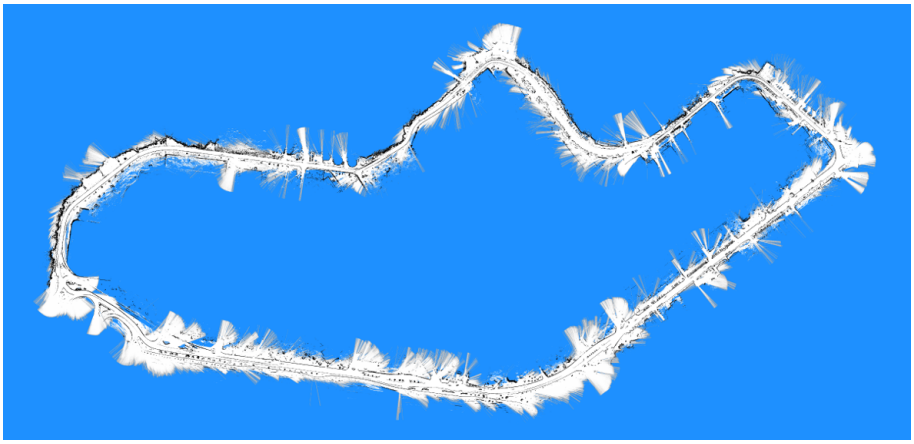


Figure 24 – Occupancy Grid Map - UB2

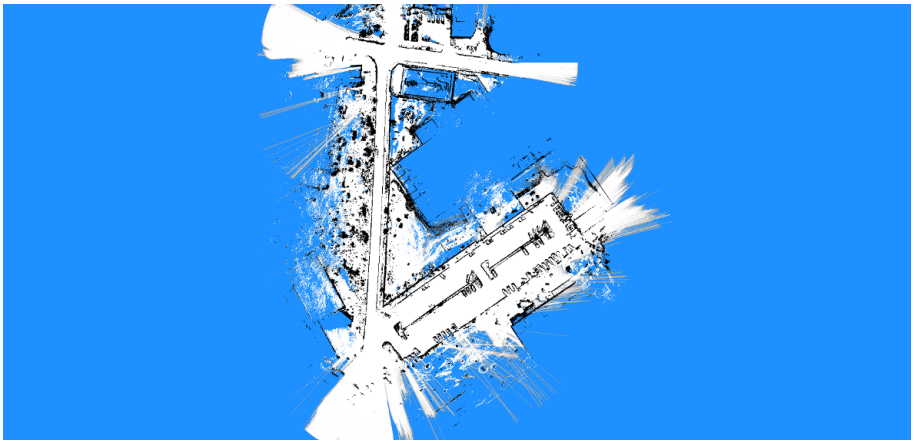


Figure 25 – Occupancy Grid Map - PL1

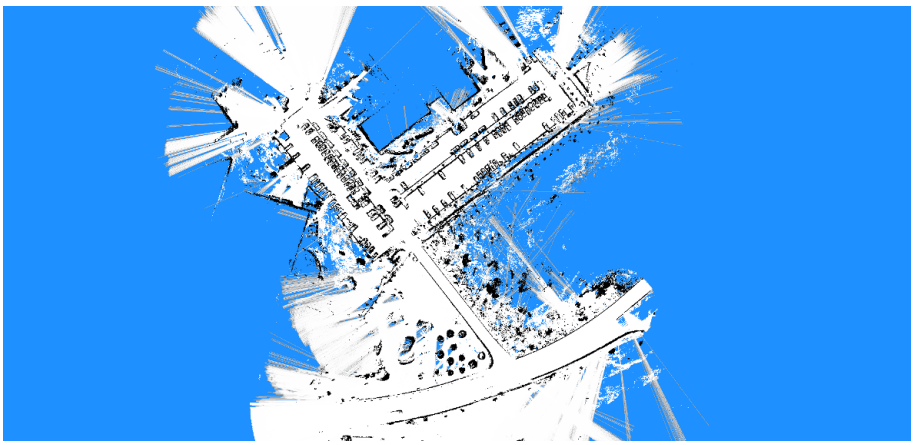


Figure 26 – Occupancy Grid Map - PL2

used to build the map in Fig. 30(c).

Fig. 31 compares the old and the new log-odds update functions in the mapping process. The left image merges the two beltway logs with the usual update function 4.27 and the right image uses 4.35. The red and the green colors represents the obstacles found respectively in the UB1 and UB2 logs. First, UB1 was used to build an intermediate map

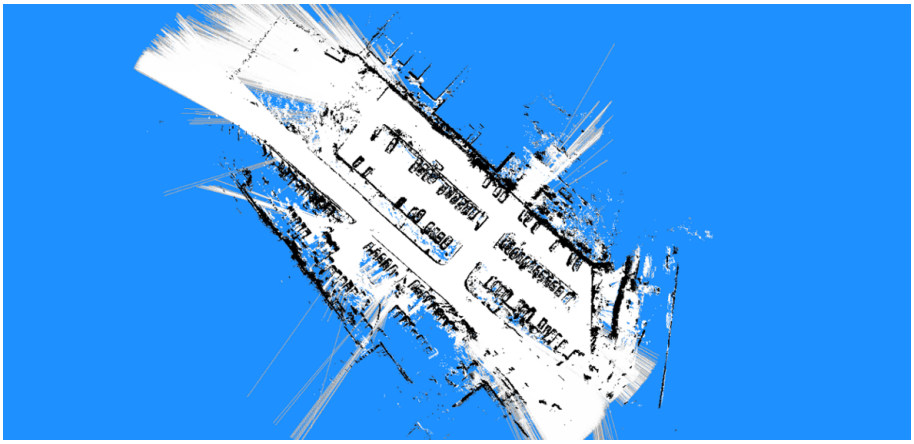


Figure 27 – Occupancy Grid Map - PL3

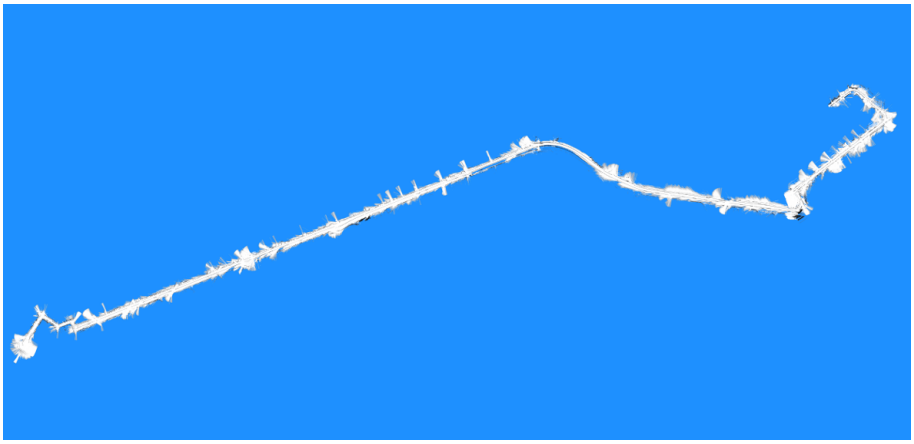


Figure 28 – Occupancy Grid Map - TRVL

which was later updated using the UB2 log. We can see the benefits of the new update function in this particular case. The most noticeable error in Fig. 31(a) and absent in Fig. 31(b) is a false positive obstacle trace generated by another vehicle in the UB2 log. UB2 also contains many parked vehicles which are only visible Fig. 31(a). This difference indicates that the new update function can help in the posterior map cleaning process.

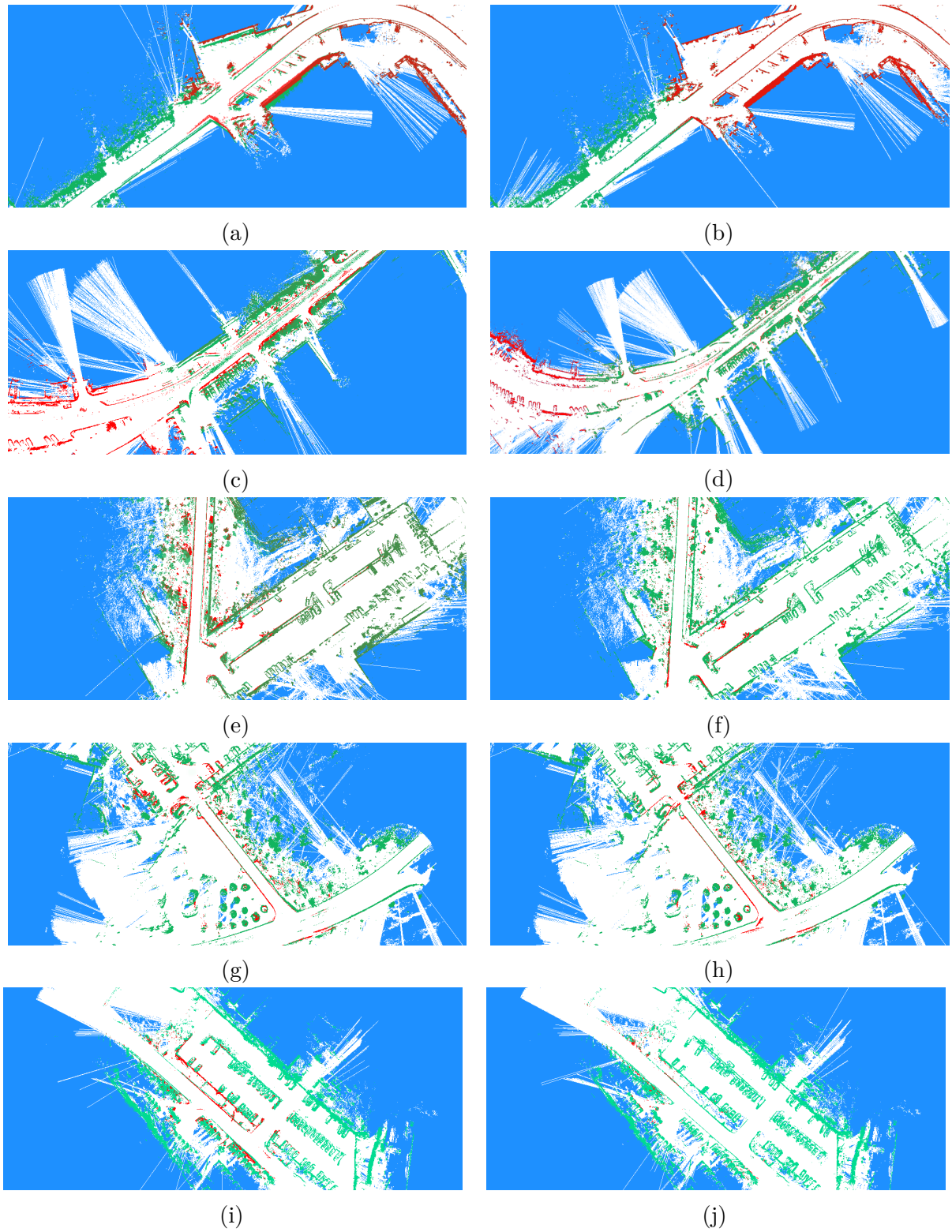
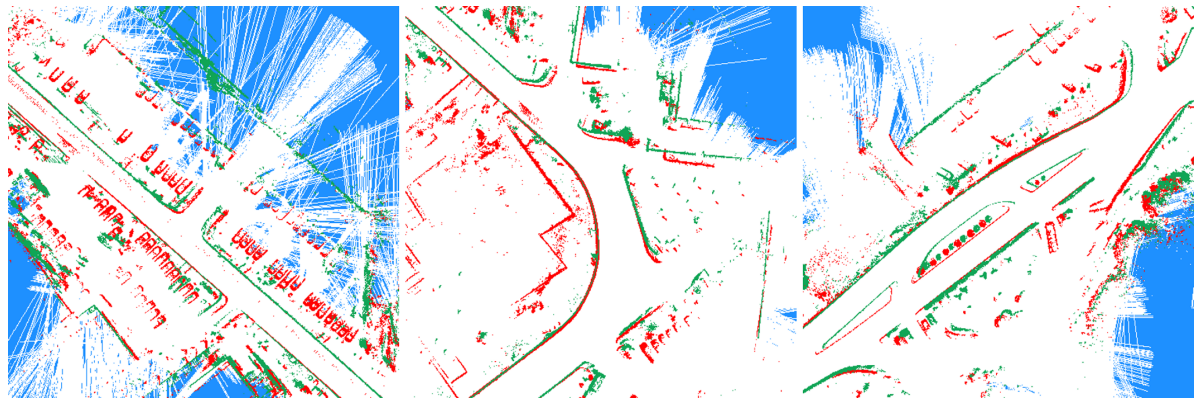
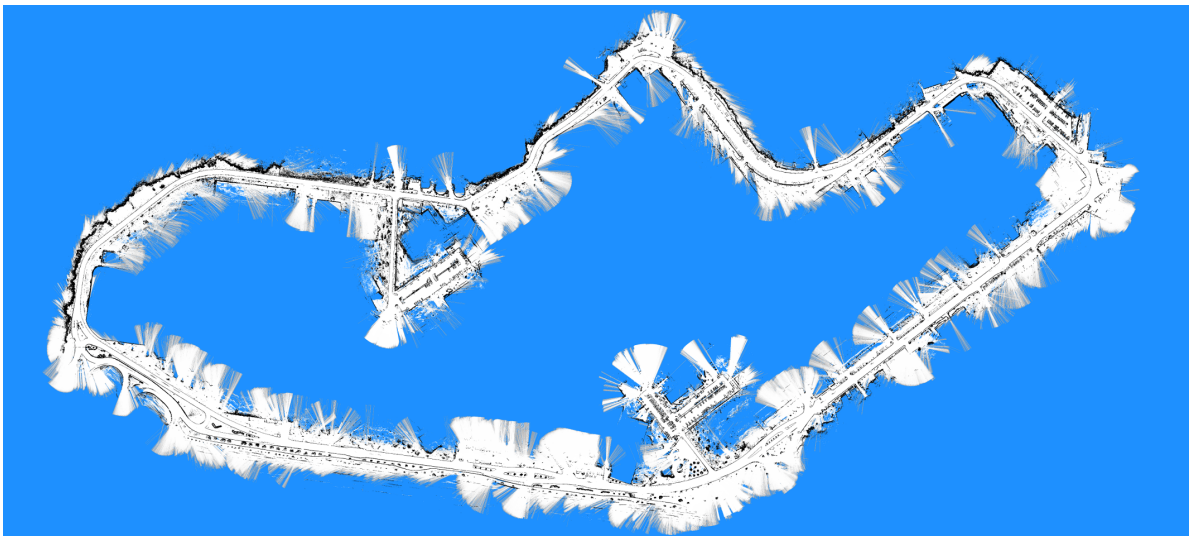


Figure 29 – Loop closure - Ablation experiments.
(a) UB1 without loop closure. (b) UB1 with loop closure. (c) UB2 without loop closure. (d) UB2 with loop closure. (e) PL1 without loop closure. (f) PL1 with loop closure. (g) PL2 without loop closure. (h) PL2 with loop closure. (i) PL3 without loop closure. (j) PL3 with loop closure.



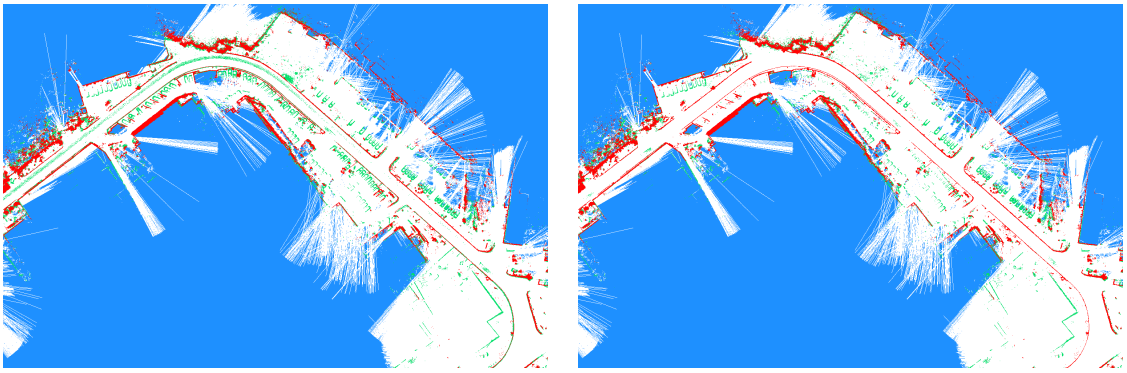
(a)

(b)



(c)

Figure 30 – Final Map.
(a) Errors in the occupancy grid map when the inter-log loop closures are disabled (UB1 and UB2). (b) The errors are drastically reduced after enabling the inter-log loop closures again (UB1 and UB2). (c) Complete map including all parking lots.



(a) (b)

Figure 31 – Comparing the log-odds update functions.

7 Conclusions

In this work, we presented a new pose graph SLAM-based mapping system for long-term map maintenance. The system uses multi-session data with the purpose of building and updating grid maps for posterior autonomous navigation and localization. The system deals with different types of sensors and can handle typical errors found in noisy sensor data. A Pre-processing module removes undesired sensor measurements from all input logs and a Hypergraph Builder module creates a hypergraph including appropriate intra-session and inter-session loop closures. A Hypergraph Optimizer module receives the hypergraph and applies several methods to estimate the AV poses from all sessions in a global coordinate frame. Along with the poses estimation, the system also makes usage of a new odometry bias calibration method in order to turn the odometry more reliable. The optimized AV poses are consumed by the Mapper module which builds the final global map. The mapper uses a custom method to temporary freeze highly observed cells in the grid maps and so block high and low dynamics in the map merging process.

We evaluated the main features of the new system using datasets collected in the vicinity of the university campus and in a specific travel to another city. The experiments indicate that the new odometry bias calibration method surpass a previous PSO-based odometry calibration available in the IARA platform. In the experiments, the system was able to jointly maximize the likelihood of the AV poses from multiple sessions in a global coordinate frame while also able to keep the estimated poses inside the GPS error limits. In a series o ablation studies, we show that the system can build the local grid maps without drifts by the virtue of the intra-session loop closures and can merge the submaps into a global map using the inter-session loop closures. The ablation also indicates that the new cell update rule applied in the map merging process can avoid undesired updates in the current map and help in posterior map cleaning process.

While defining the error in Eq. 4.7, Kümmerle et al. [31] also generate a special node $\boldsymbol{N}^s$ in the hypergraph to represent the sensor pose $\boldsymbol{s}$ and hyperedges connecting $\boldsymbol{N}^s$ to the robot poses at $\boldsymbol{N}_i$ and $\boldsymbol{N}_j$. The purpose of the new node and the new edges is to find the 2D pose of the sensor in the robot coordinate frame directly in the hypergraph optimization and the authors demonstrated good results for some laser scanner models in real world experiments. We did not obtain same quality results for the Velodyne HDL-32E in our AV, therefore we do not consider our system capable of calibrating the sensor extrinsic. In a future work, we can investigate proper methods to calibrate the extrinsic of our LiDAR and cameras.

Bibliography

[1] Cadena, Cesar, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard: *Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age.* IEEE Transactions on Robotics, 32(6):1309–1332, 2016.

[2] Badue, Claudine, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Paixão, Filipe Mutz, Lucas Veronese, Thiago Oliveira-Santos, and Alberto Ferreira De Souza: *Self-Driving Cars: A Survey.* arXiv e-prints, page arXiv:1901.04407, January 2019.

[3] Mutz, Filipe, Lucas Veronese, Thiago Oliveira-Santos, Edilson Aguiar, Fernando [Auat Cheein], and Alberto De Souza: *Large-scale mapping in complex field scenarios using an autonomous car.* Expert Systems with Applications, 46:439–462, March 2016.

[4] De Paula Veronese, L., J. Guivant, F. A. Auat Cheein, T. Oliveira-Santos, F. Mutz, E. de Aguiar, C. Badue, and A. F. De Souza: *A light-weight yet accurate localization system for autonomous cars in large-scale and complex environments.* In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 520–525, 2016.

[5] Sarcinelli, Renan, Rânik Guidolini, Vinicius B. Cardoso, Thiago M. Paixão, Rodrigo Ferreira Berriel, Pedro Azevedo, Alberto Ferreira de Souza, Claudine Badue, and Thiago Oliveira-Santos: *Handling pedestrians in self-driving cars using image tracking and alternative path generation with frenét frames.* Comput. Graph., 84:173–184, 2019.

[6] Possatti, Lucas, Ranik Guidolini, Vinicius Cardoso, Rodrigo Berriel, Thiago Paixão, Claudine Badue, Alberto De Souza, and Thiago Oliveira-Santos: *Traffic light recognition using deep learning and prior maps for autonomous cars.* pages 1–8, July 2019.

[7] Tabelini Torres, Lucas, Thiago Paixão, Rodrigo Berriel, Alberto De Souza, Claudine Badue, Nicu Sebe, and Thiago Oliveira-Santos: *Effortless deep training for traffic sign detection using templates and arbitrary natural images.* 2019 International Joint Conference on Neural Networks (IJCNN), pages 1–7, July 2019.

[8] Cardoso, V., J. Oliveira, T. Teixeira, C. Badue, F. Mutz, T. Oliveira-Santos, L. Veronese, and A. F. De Souza: *A model-predictive motion planner for the iara au-*

tonomous car. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 225–230, 2017.

[9] Guidolini, R., C. Badue, M. Berger, L. d. P. Veronese, and A. F. De Souza: *A simple yet effective obstacle avoider for the iara autonomous car.* In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1914–1919, 2016.

[10] Guidolini, R., A. F. De Souza, F. Mutz, and C. Badue: *Neural-based model predictive control for tackling steering delays of autonomous cars.* In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4324–4331, 2017.

[11] *Multiple-Robot Simultaneous Localization and Mapping: A Review.* Journal of Field Robotics, 33(1):3–46, jan 2016.

[12] Bresson, Guillaume, Zayed Alsayed, Li Yu, and Sebastien Glaser: *Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving.* IEEE Transactions on Intelligent Vehicles, 2(3):194–220, 2017.

[13] Deutsch, Isaac, Ming Liu, and Roland Siegwart: *A framework for multi-robot pose graph SLAM.* 2016 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2016, pages 567–572, 2016.

[14] Sun, Yong, Rongchuan Sun, Shumei Yu, and Yan Peng: *A Grid Map Fusion Algorithm Based on Maximum Common Subgraph.* In *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, volume 2018-July, pages 58–63. IEEE, jul 2018.

[15] Schuster, Martin J., Christoph Brand, Heiko Hirschmuller, Michael Suppa, and Michael Beetz: *Multi-robot 6D graph SLAM connecting decoupled local reference filters.* IEEE International Conference on Intelligent Robots and Systems, 2015-Decem(Iros):5093–5100, 2015.

[16] Lazaro, M. T., L. M. Paz, P. Pinies, J. A. Castellanos, and G. Grisetti: *Multi-robot SLAM using condensed measurements.* IEEE International Conference on Intelligent Robots and Systems, pages 1069–1076, 2013.

[17] Kim, Been, Michael Kaess, Luke Fletcher, John Leonard, Abraham Bachrach, Nicholas Roy, and Seth Teller: *Multiple relative pose graphs for robust cooperative mapping.* Proceedings - IEEE International Conference on Robotics and Automation, pages 3185–3192, 2010.

[18] Mangelson, Joshua G., Derrick Dominic, Ryan M. Eustice, and Ram Vasudevan: *Pairwise Consistent Measurement Set Maximization for Robust Multi-Robot Map Merging.* Proceedings - IEEE International Conference on Robotics and Automation, pages 2916–2923, 2018.

[19] Agarwal, P., G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard: *Robust map optimization using dynamic covariance scaling.* In *2013 IEEE International Conference on Robotics and Automation*, pages 62–69, 2013.

[20] Olson, E, M Walter, S Teller, and J J Leonard: *Single-cluster spectral graph partitioning for robotics applications.* In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, jun 2005.

[21] Einhorn, Erik and Horst Michael Gross: *Generic NDT mapping in dynamic environments and its application for lifelong SLAM.* Robotics and Autonomous Systems, 69:28–39, 2015.

[22] Bonanni, Taigo Maria, Bartolomeo Della Corte, and Giorgio Grisetti: *3-D Map Merging on Pose Graphs.* IEEE Robotics and Automation Letters, 2(2):1031–1038, 2017.

[23] Serafin, J. and G. Grisetti: *Nicp: Dense normal based point cloud registration.* In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 742–749, 2015.

[24] Lázaro, María T., Roberto Capobianco, and Giorgio Grisetti: *Efficient Long-term Mapping in Dynamic Environments.* IEEE International Conference on Intelligent Robots and Systems, pages 153–160, 2018.

[25] Tanaka, Kanji: *Deformable Map Matching to Handle Uncertain Loop-Less Maps.* Journal of Advanced Computational Intelligence and Intelligent Informatics, 22(6):915–923, oct 2018.

[26] Jiang, Binqian, Yilong Zhu, and Ming Liu: *A triangle feature based map-to-map matching and loop closure for 2D graph SLAM.* IEEE International Conference on Robotics and Biomimetics, ROBIO 2019, pages 2719–2725, December 2019.

[27] Ding, Xiaqing, Yue Wang, Huan Yin, Li Tang, and Rong Xiong: *Multi-session map construction in outdoor dynamic environment.* 2018 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2018, pages 384–389, 2019.

[28] Thrun, Sebastian and Michael Montemerlo: *The graph SLAM algorithm with applications to large-scale mapping of urban structures.* International Journal of Robotics Research, 25(5-6):403–429, may 2006.

[29] Uhlman, Jeffrey: *Dynamic map building and localization for autonomous vehicles.* PhD thesis, University of Oxford, 1994.

[30] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[31] Kümmerle, Rainer, Giorgio Grisetti, and Wolfram Burgard: *Simultaneous parameter calibration, localization, and mapping.* Advanced Robotics, 26(17):2021–2041, 2012.

[32] Smith, Randall, Matthew Self, and Peter Cheeseman: *Estimating uncertain spatial relationships in robotics.* In *Cox I.J., Wilfong G.T. (eds) Autonomous Robot Vehicles*, volume 1, pages 167–193. Springer-Verlag, July 1990.

[33] Bergman, Walter: *The basic nature of vehicle understeer-oversteer.* SAE Transactions, 74:387–422, 1966.

[34] Segal, Aleksandr, Dirk Hähnel, and Sebastian Thrun: *Generalized-icp.* June 2009.

[35] Almeida, J. and V. M. Santos: *Real time egomotion of a nonholonomic vehicle using lidar measurements.* J. Field Robotics, 30(1):129–141, 2013.

[36] Józsa, O., A. Börcs, and Csaba Benedek: *Towards 4d virtual city reconstruction from lidar point cloud sequences.* ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, II-3/W1:15–20, May 2013.

[37] Thrun, Sebastian and John J. Leonard: *Simultaneous Localization and Mapping*, pages 871–889. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[38] Montemerlo, M., N. Roy, and S. Thrun: *Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit.* In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2436–2441, 2003.