Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Μηχανολόγων Μηχανικών

Τομέας Μηχανολογικών Κατασκευών και Αυτομάτου Ελέγχου

Εργαστήριο Αυτομάτου Ελέγχου και Ρύθμισης Μηχανών και Εγκαταστάσεων

Διπλωματική Εργασία

Σχεδιασμός και Ανάπτυξη Συστήματος SLAM για την ταυτόχρονη χαρτογράφηση και προσδιορισμό θέσης με πολλαπλά τροχοφόρα ρομπότ.

Συγγραφέας:
Νίκολαος Κούκης
Α.Μ: 02111068

Επιβλέπων:
Καθηγητής Κωνσταντίνος Κυριακόπουλος
Επιβλέπων:
Δρ. Γεώργιος Καρράς

Ιούνιος 2017

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Κώστα Κυριακόπουλο, για την πολύπλευρη συνεισφορά του καθ' όλη την διάρκεια της διπλωματικής καθώς και για την ευκαιρία που μου έδωσε να είμαι μέλος της ομάδας του CSL.

Ευχαριστώ τον Γιώργο Καρρά για την πολύτιμη βοήθεια αλλά και για την συνολική επίβλεψη αυτής της διπλωματικής εργασίας καθώς και τους Σαχάμπ Χεσμάτι, Πάνο Μαράντο και Χαράλαμπο Μπεχλιούλη για τις ενδιαφέρουσες συζητήσεις και συμβουλές τους.

Ευχαριστώ τα μέλη του εργαστηρίου του CSL (και πλέον φίλους μου) Γιώργο, Χρήστο, Μιχάλη, Κώστα, Παναγιώτη, Μπάμπη.

Ευχαριστώ επίσης τους φίλους μου Χρήστο, Κώστα, Πάνο, Άγγελο για την καθημερινή ενθάρρυνση και υποστήριξη τους.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου Λεωνίδα και Γιούλα και τα αδέλφια μου Παναγιώτη, Χρήστο και Βαγγέλη για όλα όσα μου έχουν προσφέρει μέχρι σήμερα. Τίποτα από όλα αυτά δεν θα ήταν εφικτό χωρίς εσάς. Ευχαριστώ.

Περίληψη

Η παρούσα έκθεση αποτελεί την αναφορά της διπλωματικής εργασίας με θέμα "Σχεδιασμός και ανάπτυξη συστήματος SLAM για την ταυτόχρονη χαρτογράφηση και προσδιορισμό θέσης σε πολλαπλά τροχοφόρα ρομπότ". Κατά τη διάρκεια αυτής, προχωρήσαμε στα παρακάτω:

- Μελέτη και σύγκριση βασικών αλγορίθμων SLAM (με ένα ή πολλαπλά ρομποτ) με τελικό σκοπό την επιλογή της καλύτερης με βάση τις συνθήκες υπό τις οποίες θα εκτελέσουμε SLAM. Η μέθοδος SLAM ενός ρομποτ που επιλέχθηκε και τελικά υλοποιήθηκε με επιτυχία είναι το graphSLAM.
- Ανάπτυξη αλγορίθμων graphSLAM ενός ρομποτ μέσω του λογισμικού ρομποτικής MRPT[1]. Συμπληρωματικά υλοποιήθηκαν επεκτάσεις αυτού με χρήση του λογισμικού ROS[2], για online χρήση των αλγορίθμων (εκτέλεση του αλγορίθμου παράλληλα με την απόκτηση των δεδομένων από τους αισθητήρες). Ο κώδικας που αναπτύχθηκε διατίθενται ως ανοιχτό λογισμικό (open-source) στην σελίδα του MRPT και μέσω του github[3].
- Υλοποίηση συστήματος επικοινωνίας (ad-hoc) μεταξύ τροχοφόρων ρομπότ για την αποτελεσματική ανταλλαγή μετρήσεων/δεδομένων και την συνεργατική εκτέλεση SLAM.
- Υλοποίηση αλγορίθμου για την συνεργατική χαρτογράφηση και προσδιορισμό θέσης πολλαπλών τροχοφόρων ρομπότ (Multi-robot SLAM).
- Ενδελεχής έλεγχος των αναπτυσσόμενων αλγορίθμων στο περιβάλλον προσομοίωσης Gazebo[4] καθώς και σε πραγματικές συνθήκες μέσω πειραμάτων που πραγματοποιήθηκαν με ένα και με περισσότερα ρομποτ.

[1]`http://www.mrpt.org`
[2]`http://www.ros.org`
[3]`https://github.com/bergercookie/mrpt`
[4]`http://gazebosim.org`

National Technical University of Athens

School of Mechanical Engineering

Section of Mechanical Design and Automatic Control

Control Systems Laboratory

Diploma Thesis

Design and Development of Single and Multi-Robot Simultaneous Localization and Mapping (SLAM) Algorithms

Author:
Nikolaos Koukis
UIN: 02111068

Supervisor:
Professor Kostantinos Kyriakopoulos
Supervisor:
Dr. George Karras

June 2017

Contents

List of Figures

Listings

List of Algorithms

Nomenclature

EIF	Extended Information Filter
EKF	Extended Kalman Filter
ERD	Edge Registration Decider
GRV	Gaussian Random Variable
GSO	GraphSLAM Optimizer
IF	Information Filter
KF	Kalman Filtering
LC	Loop Closure
MR-SLAM	Multi-Robot SLAM
MRPT	Mobile Robot Programming Toolkit
NRD	Node Registration Decider
PF	Particle Filtering
RB	Rao-Blackwellisation
ROS	Robot Operating System
SEIF	Sparse Extended Information Filter
SLAM	Simultaneous Localization and Mapping
SR-SLAM	Single-Robot SLAM
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UKF	Unscented Kalman Filter
UT	Unscented Transform
UUV	Unmanned Underwater Vehicle

Chapter 1

Introduction

An autonomous robot needs to address two critical problems to survive and navigate within its surroundings: mapping the environment and finding its relative location within the map. Simultaneous localization and mapping (SLAM) is a process that aims to localize an autonomous mobile robot in a previously unexplored environment while constructing a consistent and incremental map of its environment [3]. While filtering methods (extended Kalman filtering, information-form filtering, particle filtering) used to dominate the SLAM literature, recently (~2006) graph-based approaches have made a comeback. Introduced by Lu and Milios in 1997 [4] graph-based approaches formulate SLAM as a least-squares minimization problem.

In the thesis at hand, we designed and implemented a *generic algorithm* for executing graphSLAM in the MRPT[1] robotics toolkit for single and multi-robot cases. The algorithm has been tested in both, simulated datasets (generated either by the GridmapNavSimul[2] tool or in the Gazebo[3] simulator) and online setups for both the single and multi-robot case. Wrapper code is also available for running graphSLAM as a native ROS[4] application.

In the following lines we offer a brief preview of the contents of the upcoming sections. In sec. 1.1 we outline common situations and problems in which single as well as multi-robot SLAM is often deployed. In sec. 2 we begin with a historical overview of robotic mapping and SLAM, and we introduce the major SLAM algorithms that have been used so far, paying special attention to the graphSLAM-related algorithms (sec. 2.1.4). We then move on to the design and implementation of the single-robot part of our algorithm, sec. 3, the communication strategy for multiple robot agents to exchange data, sec. 4, and the extension of the single-robot algorithms to multiple agents in sec. 5. In sec. 6 we draw conclusions and describe future directions and thoughts based on the work we conducted. Finally we provide appendices that give insight into their respective subjects.

The LaTeXsource code for this report is accessible here[5]

Keywords --- SLAM, graphSLAM, loop-closure, multi-robot, MRPT, ROS, Gazebo

[1]`www.mrpt.org`

[2]`http://www.mrpt.org/list-of-mrpt-apps/application_gridmapnavsimul/`

[3]`http://www.gazebosim.org`

[4]`www.ros.org`

[5]`controlsystemslab.gr/code/bergercookie/mr-slam-thesis-text`

1.1 SLAM common applications and usages

Having a map of the environment and an accurate estimation of the robot's trajectory, is a basic prerequisite for a variety of tasks in robotics (e.g., navigation, object manipulation - interaction with environment). During the past decade SLAM algorithms have been extensively utilized in the robotics research or in consumer applications. Among others, SLAM has successfully been used in areas such as:

- Self-driving cars - Unmanned Ground Vehicles (UGV)
- Unmanned aerial vehicles (UAV)
- Underwater vehicles (UUV)
- Planetary rovers
- Domestic robots
- Biomedical applications

Below is a list of notable cases in which SLAM algorithms have been employed, either as the central piece or as a supporting one.

- Autonomous (self-driving) cars is a standard field in which SLAM algorithms are utilized. As an example, *STANLEY* (fig. 1.1.2), winner of the 2005 *Grand Challenge* of *DARPA* performed SLAM as part of its autonomous driving system. It essentially estimated its pose by acquiring information from various onboard sensors and feeding the latter to a UKF filter, while simultaneously, using 5 SICK laser range finders, built a map of the surrounding environment as well as non-drivable objects. Using its estimated pose and map it could confidently execute obstacle avoidance and navigate towards its goal [5].
- *iRobot*'s cleaning robot, "*Roomba*" (fig. 1.1.1(a)) has integrated visual SLAM to efficiently traverse the floor to be cleaned, as well as navigate back to its charging station. On the other hand, *Neato*'s cleaning robots (fig. 1.1.1(b)) also execute SLAM, by utilizing information from an onboard laser range finder.
- To explore and map large portions of the planet surface in Mars, NASA has employed SLAM algorithm variants in its Planetary rovers (fig. 1.1.3).
- Instead of focusing on autonomous navigation, *CSIRO* (Australia's national science agency) developed GeoSLAM[6], a 3D SLAM algorithm primarily focused on accurate 3D measurement and mapping of the environment.

[6]http://www.geoslam.com/company/about-slam/

(a) The *Roomba* cleaning robot. Latest models (most notably the *Roomba 900*) utilize visual SLAM techniques to build a map of their environment.

(b) The *Neato* cleaning robot. Utilizes information from an onboard laser range finder to execute SLAM

Figure 1.1.1: SLAM has recently risen to popularity in domestic applications such as cleaning

Figure 1.1.2: The *STANLEY* self-driving car, winner of 2005 DARPA, used SLAM as part of its driving system.

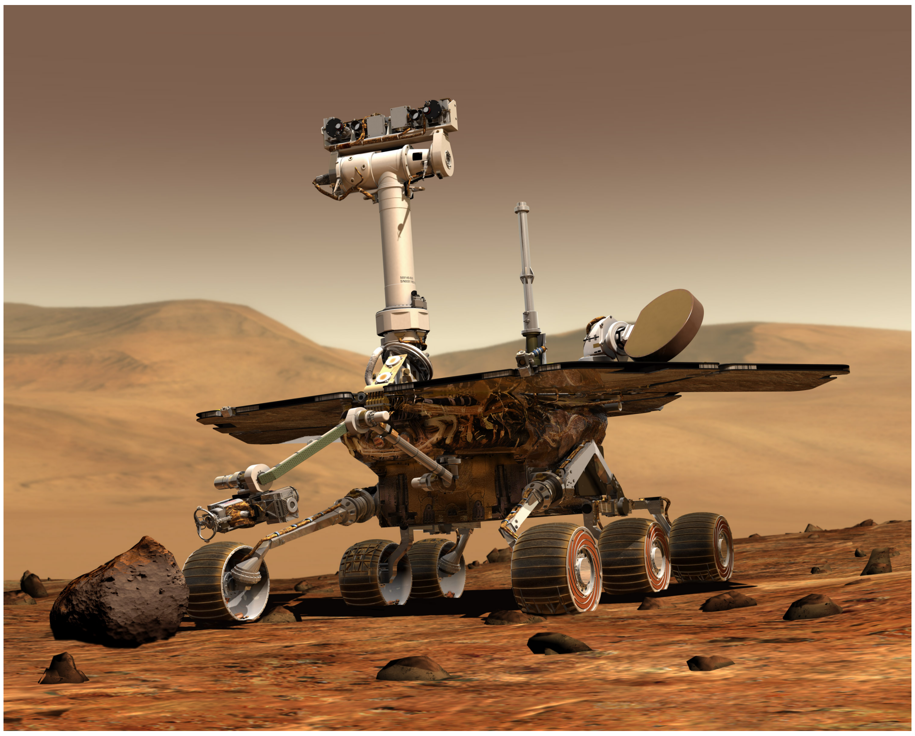


Figure 1.1.3: NASA Planetary rovers use SLAM to map the planet surface of mars.

Furthermore, using *multiple* robot agents in the mapping procedure instead of a single one offers advantages, most notably:

Reduction in needed time
Since multiple agents are concurrently mapping the environment, the procedure as a whole will finish in less time.

Improvement of overall accuracy
Since agents' gathered information and generated maps have a certain overlap, one agent could potentially improve certain distortions in its own map by utilizing information sent by its neighbors, as implemented in [6].

Decentralization of SLAM Procedure
SLAM procedure is not entirely dependent on the performance of a single agent and can be completed successfully even if one (or several) of the agents does not execute mapping appropriately or a malfunction is caused. This adds to the overall robustness and fault-tolerance of the mapping procedure.

Heterogeneity in group Multi-robot SLAM is not constrained in using robots of the same type. As a result, heterogeneity in a group could assist in traversing larger portion of the environment. For example, if we consider a combination of a ground vehicle and a quadcopter, we notice that the quadcopter can scan and traverse different height levels (not constraint by e.g., stairs, different floors) while the ground vehicle on the other hand can traverse and map narrow crossings, or low ceilings at which the quadcopter can't go to.

Chapter 2

Literature Review

2.1 Single-Robot SLAM

Due to the broad range of algorithms as well as applications in which SLAM has been successfully employed, there are two criteria with which one could categorize the various SLAM implementations [7]. These are outlined below:

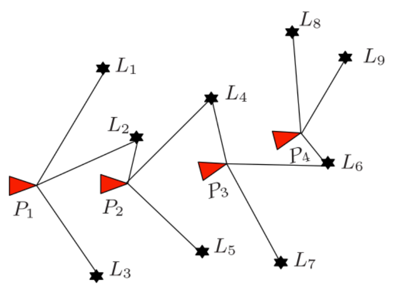
Type of map representation

Feature-based SLAM
Artificial features either preexist (e.g., cylindrical obstacles of fixed radius) or are extracted from raw sensor measurements. First SLAM implementations made use of this technique to construct a *landmarks* map. Even though this strategy is fairly lightweight with regards to its computational cost, it can only be used in indoor environments where certain discrete features can be extracted (e.g., wall corners, edges). Furthermore, in this map representation (fig. 2.1.1(a)), information is lost prior to processing, since the SLAM algorithm operates on the extracted landmarks only.

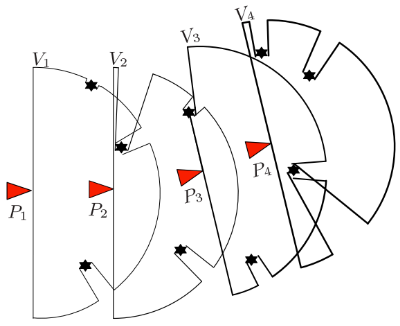
View-based SLAM
Raw sensor measurements are used to produce occupancy grid maps. In these kinds of maps, the traversed environment is split into a grid of cells in which each cell is assigned a number that corresponds to the possibility that it is occupied by an obstacle. To build this kind of maps (fig. 2.1.1(b)) we often employ laser scanners. Even though this map representation often demands higher storage cost as well as more time to process, the generated maps are of much higher detail than the ones generated by feature-based techniques.

Appearance-based SLAM
Appearance-based SLAM 2.1.2 is an effective method for the loop closure problem and can also be used in combination with feature-based or view-based techniques. In this method, new observations, either features or views, are matched against available reference observations to identify a previously observed location. (For instance, the ATLAS framework [8] provides occupancy grid maps from outdoor urban environments using a laser range scanner mounted on top of a car).

Polygon-based SLAM
In this method, planar segments, composed of infinite planes, are associated with features of the environment. In contrast to the rest of the presented map



(a) Individual landmarks from the environment are extracted and are used for SLAM



(b) View-based map representation. Sensors that can associate the entire measurements from different positions are employed (e.g., laser scanners)

Figure 2.1.1: Examples of 4 poses with the corresponding representation of their environment

representations, the advantage of the polygon-based SLAM is that it produces highly detailed small-sized maps and is well-suited for higher-level tasks such as interacting with the environment; however, the process of fitting planar segments creates extra computational demand.

Data processing scheme
Comprises the main differentiation criterion. For that case these techniques are rigorously analysed in the rest of this section.

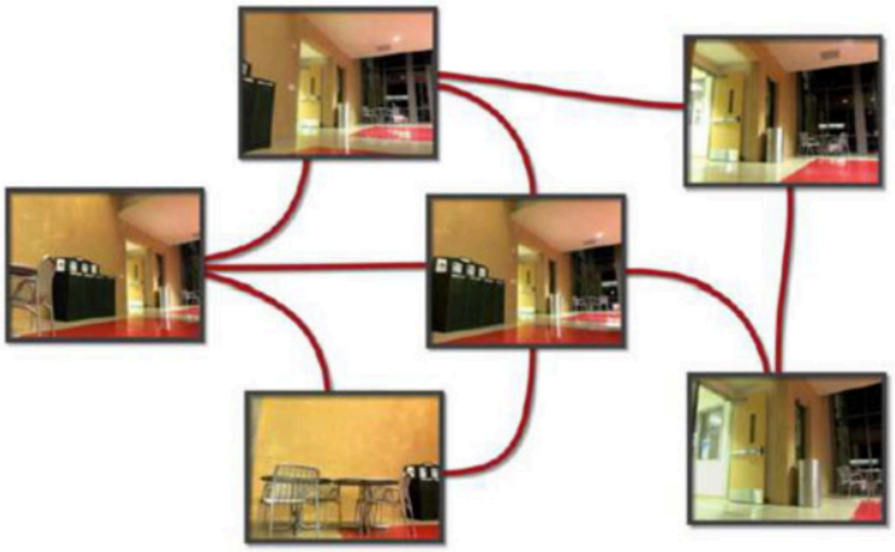


Figure 2.1.2: Example of an appearance-based slam representation, where the robot stores an image for each node of a graph and graph edges connect similarly looking images [9].

The following is an overview of the existing methodologies that have been successfully employed in single-robot SLAM situations. At first an overview of the Kalman filter and its variants is presented (sec. 2.1.1). In sec. 2.1.2 particle-filtering techniques are examined while finally extra attention is given to the variant of choice, that is the graph-based strategies. (sec. 2.1.4).

2.1.1 KF-SLAM

Formulation of SLAM in Bayesian representation (as is utilized in the Kalman Filter) is provided in the appendix (appendix A), where the necessary terminology is also defined. Current section contains an overview of the Kalman Filter-related techniques to solving the SLAM problem. Notable works on this field include the following:

Extended Kalman Filter (EKF) [10]

The EKF constitutes the first attempt in handling both landmarks and the uncertainty in robot movement *simultaneously*, by handling both as states in a Kalman-filter formulation. The novelty resided in the extended term, since the non-linearities of both the movement and observation models were handled by a prior linearisation step.

The following analysis is based on the work in [11]. An excellent example of a simple EKF implementation along with the corresponding MATLAB code is available in [12].

Solving SLAM in the Bayesian context (as in EKF-SLAM) depends on finding appropriate expressions for the motion and observation models. First step to the EKF method is to formulate the motion and observation models in suitable forms so that the Kalman filter can handle them. In eqn. 2.1.1 $\boldsymbol{f}(\cdot)$ models the vehicle kinematics, while $\boldsymbol{w}_k$ represents the additive, zero-mean uncorrelated Gaussian motion disturbances with covariance $\boldsymbol{Q}_k$.

$$P(\boldsymbol{x}_k|\boldsymbol{x}_{k_1}, \boldsymbol{u}_k) \Longleftrightarrow \boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{2.1.1}$$

The observation model is defined as follows:

$$P(\boldsymbol{z}_k|\boldsymbol{x}_{k_1}, \boldsymbol{m}) \Longleftrightarrow \boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}) + \boldsymbol{v}_k, \tag{2.1.2}$$

where $\boldsymbol{h}(\cdot)$ describes the geometry of observation, while $\boldsymbol{v}_k$ are the additive zero-mean uncorrelated Gaussian observation disturbances with covariance $\boldsymbol{R}_k$.

Having defined the motion and observation models, we use the following formulas to compute the mean and covariance for the latter joint posterior distribution $P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)$:

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}} \end{bmatrix} = \boldsymbol{E} \begin{bmatrix} \boldsymbol{x}_k \\ \hat{\boldsymbol{m}}_k \end{bmatrix} | \boldsymbol{Z}_{0:k} \Bigg] \tag{2.1.3}$$

$$\begin{aligned} \boldsymbol{P}_{k|k} &= \begin{bmatrix} \boldsymbol{P}_{xx} & \boldsymbol{P}_{xm} \\ \boldsymbol{P}_{xm}^T & \boldsymbol{P}_{mm} \end{bmatrix}_{k|k} \\ &= \boldsymbol{E} \left[\begin{pmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \\ \boldsymbol{m} - \hat{\boldsymbol{m}}_k \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \\ \boldsymbol{m} - \hat{\boldsymbol{m}}_k \end{pmatrix}^T | \boldsymbol{Z}_{0:k} \right] \end{aligned} \tag{2.1.4}$$

We also have the following equations, which are used to model the time and observation update steps of the EKF algorithm:

Time update

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k) \tag{2.1.5}$$
$$\boldsymbol{P}_{xx,k|k-1} = \nabla \boldsymbol{f} \boldsymbol{P}_{xx,k-1|k-1} \nabla f^T + \boldsymbol{Q}_k \tag{2.1.6}$$

where $\nabla \boldsymbol{f}$ is the Jacobian evaluated at the estimate $\hat{\boldsymbol{x}}_{k-1|k-1}$. Notice that in there is no need to perform this update for *stationary* landmarks[11].

Observation update

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}_{k|k-1} \\ \hat{\boldsymbol{m}}_{k-1} \end{bmatrix} + \boldsymbol{W}_k\big(\boldsymbol{z}(k) - \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1})\big) \tag{2.1.7}$$
$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{W}_k \boldsymbol{S}_k \boldsymbol{W}_k^T \tag{2.1.8}$$
$$where$$
$$\boldsymbol{S}_k = \nabla \boldsymbol{h} \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T + \boldsymbol{R}_k \tag{2.1.9}$$
$$\boldsymbol{W}_k = \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T \boldsymbol{S}_k^{-1} \tag{2.1.10}$$

where $\nabla \boldsymbol{h}$ is the Jacobian evaluated at the estimate $\hat{\boldsymbol{x}}_{k|k-1}$ and $\hat{\boldsymbol{m}}_{k-1}$.

Finally, when it comes to using the EKF in SLAM applications, the following remarks should be made:

Convergence

Convergence of the position in the estimation of landmarks (map) is determined on the monotonic convergence of the:

- Determinant of the covariance matrix $\boldsymbol{P}_{mm,k}$
- Determinant of all landmark pair submatrices towards 0.

Computational Effort

In static environments, the motion update step affects only the current position estimate (and not the map). However the observation step requires that all landmark means and covariances be computed. In the simple EKF implementation, this makes the computational and storage cost quadratic to the total number of landmarks N

Non-linearity

Even though KF-SLAM is the optimal state estimator when applied to linear models, this doesn't hold for its non-linear counterpart, the EKF.EKF-SLAM employs first-order Taylor linearisation to the non-linear motion and observation models. As discussed in [13] however, given highly non-linear models this may lead to inevitable and sometimes dramatic inconsistencies in the computed solutions.

Extended Information Filter (EIF)

Even though the *information* and standard *covariance* representation of the SLAM problem are equivalent, by formulating it in the former, and by taking advantage of its sparse nature, Thrun et al. [14] managed to reduce the computation time significantly. A sparse information matrix can be stored in less memory and the sparsity allows much faster recovery of the posterior mean and covariance [15]. Only drawback to this, from a theoretical point of view, was that the solution was merely an approximation and not a precise solution to the problem. Sparsity of the information matrix was achieved via truncating non-diagonal elements of it, which can lead to overconfident error bounds[16]. More specifically, while the state estimates are only slightly overconfident when expressed in a local reference frame, they suffer from an exaggerated global inconsistency [17].

Compressed Extended Kalman Filter (CEKF) [18] [19]

The CEKF is a method initially introduced in [18] and extended in [19] by adding a data association step, the *Relative Landmark Representation* (RLR). Its goal is to reduce EKF SLAM requirements, both in terms of computational complexity as well as memory. More specifically, by constraining the SLAM procedure in smaller areas which contain N_α landmarks ($N_\alpha \ll N$, where N is the total amount of landmarks in the robot's global map), CEKF reduces the complexity of computations from $\mathcal{O}(N^2)$ to $\mathcal{O}(N_\alpha^2)$. However, this doesn't hold in the situations that the robot is transitioning from one local area to another where a full SLAM update is required, thus $\mathcal{O}(N^2)$. The latter corner case (which may be important when these transitions are frequent) is dealt with by taking in account only the active/local states when executing the full SLAM update. This essentially reduces the full SLAM update complexity to $\mathcal{O}(N \times N_b)$, where N_b is the number of landmarks in the active/local states (active landmarks). The use of active states also reduces the memory requirements of the procedure to $N \times N_b$ [19].

Unscented Kalman Filter [1]

Despite the de facto solution for estimation/tracking problems, the EKF filter, due to its handling of non-linearities (that is, linearisation around the current estimate), is a reliable and robust solution only in situations that these non-linearities are mild. UKF comprises a linear estimator which, instead of linearising the model dynamics, uses *a set of discretely sampled points* that can appropriately *parameterize the mean and covariance* [20].

To do that, the UKF utilizes the *Unscented Transform*, introduced in [20]. The unscented transformation (UT) is a method for calculating the statistics of a random variable which undergoes a non-linear transformation: Consider propagating a random variable $\boldsymbol{x} \in \mathbb{R}^L$ through a non-linear function $\boldsymbol{y} = g(\boldsymbol{x})$. Also assume that the mean and covariance of $\boldsymbol{x}$ are $\bar{\boldsymbol{x}}$ and $\boldsymbol{P_x}$ respectively. If we were to use the EKF, $\boldsymbol{y}$ would be approximated analytically by propagating $\boldsymbol{x}$ through the first-order linearised model of g. Instead, UT calculates the statistics of $\boldsymbol{y}$ by forming a matrix $\boldsymbol{X}$ of $2L+1$ *sigma* vectors $\boldsymbol{X_i}$, along with their respective weights $\boldsymbol{W_i}$. These are calculated according to the following equations:

$$
\begin{aligned}
\boldsymbol{X}_0 &= \bar{\boldsymbol{x}} && (2.1.11)\\
\boldsymbol{X}_i &= \boldsymbol{x} + \left(\sqrt{L+\lambda)\boldsymbol{P}_{\boldsymbol{x}}}\right)_i,\ i = 1, \cdots, L && (2.1.12)\\
\boldsymbol{X}_i &= \boldsymbol{x} + \left(\sqrt{L-\lambda)\boldsymbol{P}_{\boldsymbol{x}}}\right)_{i-L},\ i = L+1, \cdots, 2L && (2.1.13)\\
\boldsymbol{W}_0^{(m)} &= \frac{\lambda}{(L+\lambda)} && (2.1.14)\\
\boldsymbol{W}_0^{(c)} &= \frac{\lambda}{(L+\lambda)} + (1-\alpha^2+\beta) && (2.1.15)\\
\boldsymbol{W}_i^{(m)} &= \boldsymbol{W}_i^{(c)} = \frac{1}{2(L+\lambda)}\ i = 1, \cdots, 2L && (2.1.16)
\end{aligned}
$$

where,

$\lambda = \alpha^2(L+\kappa) - L$ is a scaling parameter,

α determines the spread of the sigma points and is usually set to a small positive value,

κ is a secondary scaling parameter, usually set to 0

β is used to incorporate prior knowledge of the distribution of $\boldsymbol{x}$

(for Gaussian distributions $\beta = 2$ is optimal)

These sigma vectors are propagated through the following non-linear function:

$$\boldsymbol{Y}_i = g(\boldsymbol{X}_i),\ i = 0, \cdots, 2L \qquad (2.1.17)$$

The mean and covariance of $\boldsymbol{y}$ are finally approximated using a weighted sample mean and covariance of the posterior sigma points, that is:

$$
\begin{aligned}
\bar{\boldsymbol{y}} &\approx \sum_{i=0}^{2L} \boldsymbol{W}_i^{(m)} \boldsymbol{Y}_i && (2.1.18)\\
\boldsymbol{P}_{\boldsymbol{y}} &\approx \sum_{i=0}^{2L} \boldsymbol{W}_i^{(c)} (\boldsymbol{Y}_i - \bar{\boldsymbol{y}})(\boldsymbol{Y}_i - \bar{\boldsymbol{y}})^T && (2.1.19)
\end{aligned}
$$

An example of how UKF computes the mean and covariance via the Unscented Transform is also illustrated in fig. 2.1.3.

Until this point, the Unscented Transform was presented. To formulate it into a filter, UKF applies the UT recursively to estimate the *augmented SLAM state*, that is a concatenation of the original state and noise variables:

$$\boldsymbol{x}_k^{\alpha} = \left[\boldsymbol{x}_k^T\ \boldsymbol{v}_k^T\ \boldsymbol{n}_k^T\right]^T \qquad (2.1.20)$$

Notice that while the problem is of the same order as the EKF, no explicit computation of Jacobians or Hessians is needed [1].

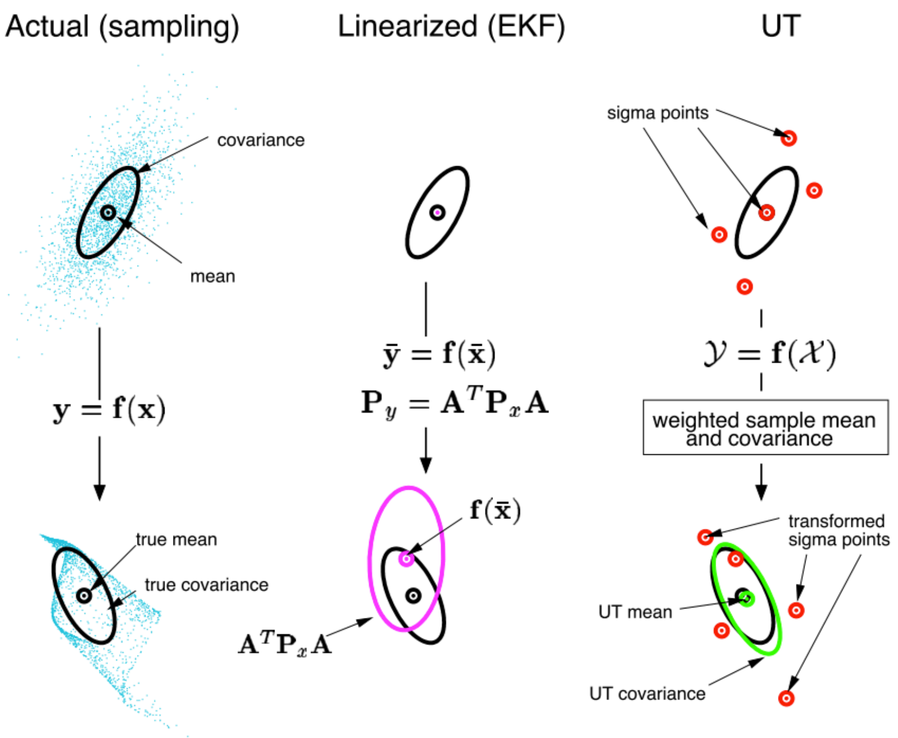
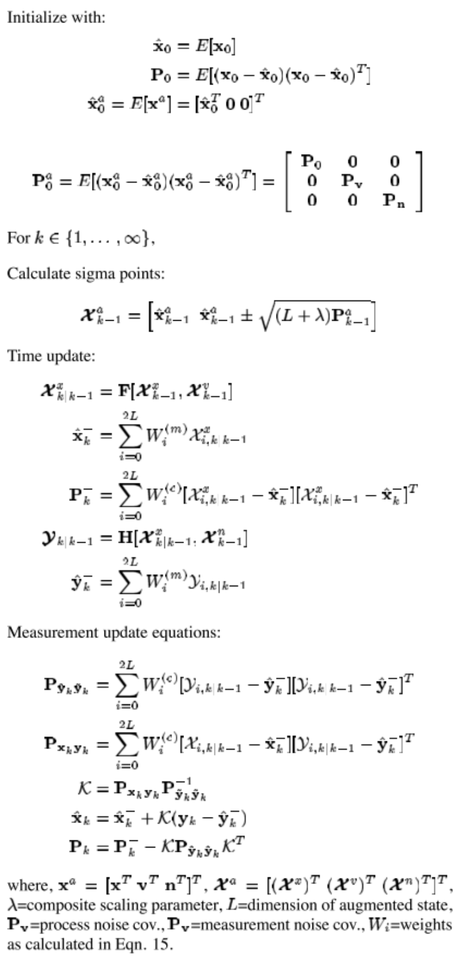

Actual (sampling)
Linearized (EKF)
UT
covariance
mean
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
true mean
true covariance
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
sigma points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

Figure 2.1.3: Example of the UT for mean and covariance propagation:
a. Actual
b. First-order linearization (EKF)
c. Unscented transform [1]



Initialize with:

$$\hat{\mathbf{x}}_0 = E[\mathbf{x}_0]$$

$$\mathbf{P}_0 = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T]$$

$$\hat{\mathbf{x}}_0^a = E[\mathbf{x}^a] = [\hat{\mathbf{x}}_0^T \; \mathbf{0} \; \mathbf{0}]^T$$

$$\mathbf{P}_0^a = E[(\mathbf{x}_0^a - \hat{\mathbf{x}}_0^a)(\mathbf{x}_0^a - \hat{\mathbf{x}}_0^a)^T] = \begin{bmatrix} \mathbf{P}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P_v} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P_n} \end{bmatrix}$$

For $k \in \{1, \ldots, \infty\}$,

Calculate sigma points:

$$\boldsymbol{\mathcal{X}}_{k-1}^a = \left[\hat{\mathbf{x}}_{k-1}^a \;\; \hat{\mathbf{x}}_{k-1}^a \pm \sqrt{(L+\lambda)\mathbf{P}_{k-1}^a}\right]$$

Time update:

$$\boldsymbol{\mathcal{X}}_{k|k-1}^x = \mathbf{F}[\boldsymbol{\mathcal{X}}_{k-1}^x, \boldsymbol{\mathcal{X}}_{k-1}^v]$$

$$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{X}_{i,k|k-1}^x$$

$$\mathbf{P}_k^- = \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-][\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-]^T$$

$$\boldsymbol{\mathcal{Y}}_{k|k-1} = \mathbf{H}[\boldsymbol{\mathcal{X}}_{k|k-1}^x, \boldsymbol{\mathcal{X}}_{k-1}^n]$$

$$\hat{\mathbf{y}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{Y}_{i,k|k-1}$$

Measurement update equations:

$$\mathbf{P}_{\tilde{\mathbf{y}}_k \tilde{\mathbf{y}}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-][\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T$$

$$\mathbf{P}_{\mathbf{x}_k \mathbf{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{X}_{i,k|k-1} - \hat{\mathbf{x}}_k^-][\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T$$

$$\mathcal{K} = \mathbf{P}_{\mathbf{x}_k \mathbf{y}_k} \mathbf{P}_{\tilde{\mathbf{y}}_k \tilde{\mathbf{y}}_k}^{-1}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathcal{K}(\mathbf{y}_k - \hat{\mathbf{y}}_k^-)$$

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathcal{K} \mathbf{P}_{\tilde{\mathbf{y}}_k \tilde{\mathbf{y}}_k} \mathcal{K}^T$$

where, $\mathbf{x}^a = [\mathbf{x}^T \; \mathbf{v}^T \; \mathbf{n}^T]^T$, $\boldsymbol{\mathcal{X}}^a = [(\boldsymbol{\mathcal{X}}^x)^T \; (\boldsymbol{\mathcal{X}}^v)^T \; (\boldsymbol{\mathcal{X}}^n)^T]^T$, λ=composite scaling parameter, L=dimension of augmented state, $\mathbf{P_v}$=process noise cov., $\mathbf{P_v}$=measurement noise cov., W_i=weights as calculated in Eqn. 15.

Figure 2.1.4: Pseudocode for the UKF algorithm [1]

2.1.2 PF-SLAM

The formulation of SLAM in Bayesian representation (as is utilized in Particle Filtering) is provided in the appendix(A), where the necessary terminology is also defined.

Particle filtering techniques rose to popularity in the SLAM scientific community with the FastSLAM [21] and FastSLAM 2.0 [22] algorithms introduced by Montemerlo et al., in 2002 and 2003 respectively. While most efforts focused on improving the performance of EKF-SLAM, while retaining its essential linear Gaussian assumptions, FastSLAM with its basis on recursive Monte Carlo sampling was the first to *directly* represent the non-linear process model and non-gaussian pose distribution [1]. However, the dimensionality of the SLAM problem still rendered the Particle-Filtering usage as computationally intractable. The novelty of FastSLAM came from the reduction of sample-space by applying Rao-Blackwellisation (RB) whereby a joint space is partitioned according to the product rule of probabilities, $P(\boldsymbol{x}_1, \boldsymbol{x}_2) = P(\boldsymbol{x}_2|\boldsymbol{x}_1)P(\boldsymbol{x}_1)$ and if $P(\boldsymbol{x}_2|\boldsymbol{x}_1)$ can be represented analytically, only $P(\boldsymbol{x_1})$ needs to be sampled $\boldsymbol{x}_1^{(i)} \sim P(\boldsymbol{x}_1)$. Therefore, the joint probability can be represented by the set $\{\boldsymbol{x}_1^{(i)}, P(\boldsymbol{x}_2|\boldsymbol{x}_1^{(i)})\}_i^N$ and we can also find the total probability of $\boldsymbol{x}_2$ by marginalising out $\boldsymbol{x}_1^{(i)}$, $\forall\, i$:

[1]Direct usage of the non-linear model was only implemented for the time update step. FastSLAM still linearises the observation model

$$P(\boldsymbol{x}_2) \approx \frac{1}{N} \sum_{i}^{N} P(\boldsymbol{x}_2 | \boldsymbol{x}_1^{(i)}) \tag{2.1.21}$$

Based on the above, the joint SLAM state may be factored into two separate probability distributions, that is a vehicle component and a conditional map component.

$$P(\boldsymbol{X}_{0:k}, \boldsymbol{m} | \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) = P(\boldsymbol{m} | \boldsymbol{X}_{0:k}, \boldsymbol{Z}_{0:k}) \times P(\boldsymbol{X}_{0:k} | \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) \tag{2.1.22}$$

Notice that in eqn. 2.1.22, the probability distribution is of the whole robot trajectory, whereas in EKF it is only on the latest robot pose. In this case, the landmark positions can be represented as independent Gaussians, because no marginalisation of robot poses is made, thus no such information is actually encoded between the estimated landmark positions. This is the key to the FastSLAM speed, since the map is now represented as a *set of independent Gaussians*, with linear complexity, rather than a joint map covariance with quadratic complexity. With regards to the trajectory, and using the RB filter, it is represented by a set of particles, each having a weight and an estimation of the entire robot trajectory.

Updating the map for the trajectory of a given particle $X_{0:k}^{(i)}$ is trivial; it is done by performing an EKF update on the observed landmarks, assuming that the robot trajectory is known. With regards to propagating the pose particles, FastSLAM utilises a series of steps which include:

Proposal distribution At each timestep for each particle a proposal distribution is drawn conditioned on the particle prior poses, observations, and latest control input.

Sample weighting Samples are weighted according to an importance function

(Optional) resampling Resampling is accomplished by selecting particles with replacement. After resampling, their weights are reset $\rightarrow w_k^{(i)} = \frac{1}{N}$, where N is the number of particles after this step.

For an in-detail analysis of the usage of RB filter as well as the FastSLAM algorithm itself, refer to [11], [21]. Notable work on PF-SLAM is also done in [23], [24], in which the authors implement a Rao-Blackwellised particle filter to build and maintain an occupancy grid map of the surrounding environment. The latter is in contrast with the strategies in *FastSLAM, FastSLAM2.0,* since Grisetti et al. do not rely on predefined landmarks and use raw laser scans to acquire the grid maps. This work is also released as open-source software and is available via the popular ROS package *gmapping*.

2.1.3 Artificial Intelligence SLAM - Topological SLAM

Artificial Intelligence has also been recently employed in solving the SLAM problem. Implementations so far are based on existing SLAM schemes (e.g. Particle filtering sec. 2.1.2) which, however, are realized using AI algorithms. These algorithms most notably include:

- Neural networks
- Fuzzy logic techniques

.

Even though not nearly as popular as EKF or Particle-Filtering, some papers exist demonstrating their potential.

- In [25] a solution based on fuzzy logic is proposed to tune the covariance values of the measurement mode.
- In [26] authors design and implement RatSLAM, a technique based on neural networks that executes SLAM by utilizing monocular camera and odometry measurements. An open-source implementation is also presented in [27].
- Semantic approaches, such as SLAM++ [28] and topological methods, which are based on the abstract information, have also been designed in the context of AI SLAM. Notable topological solutions are presented in [29] as well as [30].

A review paper outlining works in the field of topological SLAM is available in [31].

2.1.4 Graph-based approaches

A graph-based (or smoothing) approach to SLAM concerns not only the most recent robot pose, but the *entire robot trajectory* up to the current time. Optimizing over the whole robot trajectory (in the naive version) gives graph-based SLAM an advantage in terms of accuracy and loop closing capabilities against traditional filtering methods. While at a first glance this would seem to add up to the problem complexity, as more variables are added overall, it actually simplifies it. The simplification arises from the fact that the smoothing information matrix is naturally sparse, due to the incremental nature of the SLAM problem. On the contrary, the information matrix in filtering approaches becomes dense when marginalizing out previous robot poses [32]. This marginalization corresponds to the observations from robot poses to landmarks being encoded as constraints between the landmarks themselves.

When executing graphSLAM, one can consider *exclusively the robot poses* and thus execute pose-graph SLAM. This is particularly suited to sensors that are able to yield pairwise constraints between nearby robot poses [33], such as laser scanners. This offers the advantage of not being limited to generate one kind of map, as given the estimated trajectory, one can *afterwards* align the measurements with corresponding acquisition positions and build either a landmarks map or a occupancy grid map (mapping with known poses).

In graph-based SLAM approaches, the overall problem can be broken down into two subparts, a *frontend* and a *backend*:

- The frontend is responsible for constructing the initial graph (either robot-poses graph or considering both robot pose constraints and robot pose - landmark constraints) from raw sensor data. When it comes to identifying previous robot poses or already observed landmarks, the frontend also has to deal with the data association problem.
- Upon construction, the graph is passed on to the backend, which is a multivariate optimization scheme, responsible for repositioning the graph nodes so that the error vector between the predicted state and the measurements is minimized. Most backends depend on sparse versions of least squares solvers like Gauss-Newton, or the Levenberg-Marquardt scheme. A more detailed analysis of the optimisation process is provided in Stachniss' tutorial[34].

Historical overview

The following comprises an outline of the most notable work in the field of graph-based SLAM approaches [2].

- Lu and Milios [35] were the first to refine a map, by globally optimizing the system of equations to reduce the error introduced by constraints.
- Gutman and Konolige [36] proposed an effective way for constructing such a network and for detecting loop closures while running an incremental estimation algorithm.
- Dellaert and Kaess [33] were the first to exploit sparse matrix factorizations in order to solve the linearised problem in off-line SLAM. Based on this work, Kaess presented *iSAM* [32] as well as *iSAM2*[37] algorithms which use an online incremental version of the square root smoothing and mapping algorithm (SAM). iSAM algorithms take advantage of partial reordering strategies to compute the sparse factorization. iSAM is also implemented in single-robot 3D mapping applications as well as in multiple-robot, heterogeneous setup (cooperation of mobile and aerial vehicles).
- GraphSLAM [38] introduced by Thrun and Montemerlo in 2006 applies variable elimination techniques to reduce the problem dimensionality, thus reducing problem complexity.
- The *ATLAS framework*[8] works on two-levels of graphs; To build the lower level graphs a Kalman filter is used, while for the second level a global optimization framework is used to align the constructed local maps. On a similar mindset, Estrada et al. [39] uses a hierarchical approach, especially useful to reduce the dimensionality of the optimization problem when mapping larger environments.
- While the majority of previous techniques focused on the optimization scheme, Olson focused on front-end algorithms and presented an outlier rejection scheme based on spectral clustering [15]. More on Olson's work is presented in sec. 3.2 as his work in [40] was used as a guideline to create a loop closing scheme for the single robot graphSLAM algorithm.
- In terms of graph-based approaches in 3D mapping, Nüchter work [41] focuses on a front-end that finds constraints between 3D poses and utilizes a variant of Lu and Milios strategy[35] for optimizing the constructed graph.

Mathematical overview

The following comprises the mathematical formulation of a standard SLAM problem in graph-based approaches, heavily based on [34], [42]. Analysis is conducted for the pose-graph case, but is easily extensible to take discrete landmarks in account as well.

Let $x = (x_1, x_2, \cdots x_T)^T$ be a set of positions that comprise the estimated robot trajectory (i.e., graph nodes). Also let $z_{i,j}, \mathbf{\Omega}_{i,j}$ be the mean and information matrix of a *virtual measurement* that associates two different nodes of that trajectory. In pose-graph SLAM, this is the transformation from one node position to another. Such measurement can be obtained from sensors such as laser scanners by storing a measurement with each recorded

[2]Section is heavily based on the introduction section of [34]

node position and then comparing the measurements of two nodes (e.g., using the ICP algorithm) to determine a transformation $T_{i,j}$ from i to j. Given a rough estimation of those node positions we can also obtain an initial estimation of that virtual measurement - i.e., $\hat{z}_{i,j}$. Usually this prediction is the *relative transformation* between the two nodes.

Given a virtual measurement and its corresponding prediction, one can define an error multivariate function as follows:

$$e(x_i, x_j) = e_{i,j} = z_{i,j} - \hat{z}_{i,j}(x_i, x_j) \tag{2.1.23}$$

Notice that the virtual measurement doesn't depend on the x_i, x_j positions but rather on their corresponding recorded measurements. We provide an explanatory figure of the virtual measurements in fig. 2.1.5

Furthermore, the log-likelihood of the virtual measurement can be defined as follows:

$$l_{i,j} \propto \left[z_{i,j} - \hat{z}_{i,j}(x_i, x_j)\right]^T \mathbf{\Omega}_{i,j} \left[z_{i,j} - \hat{z}_{i,j}(x_i, x_j)\right] = e_{i,j}^T \mathbf{\Omega}_{i,j} \, e_{i,j} \tag{2.1.24}$$

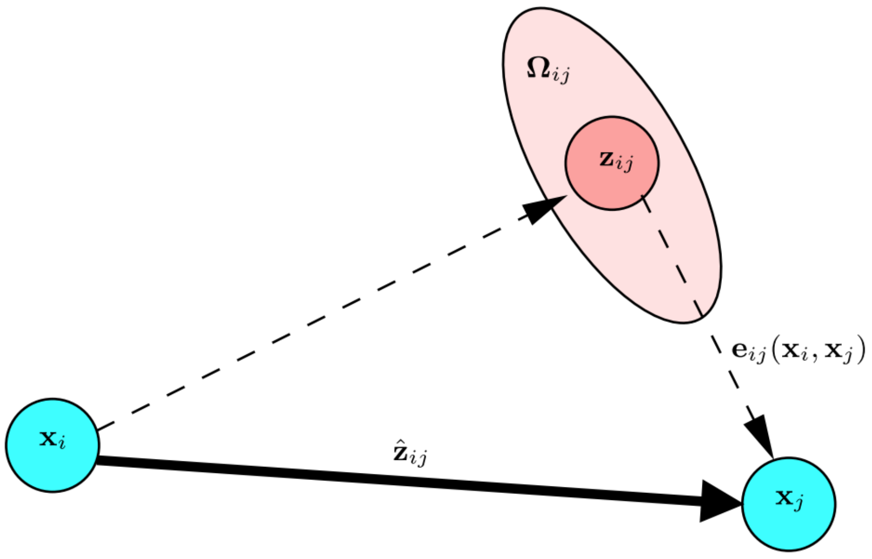


Figure 2.1.5: Representation of predicted and actual virtual measurements. Predicted value is obtained by the relative node positions prior to the optimization. Error function is defined as the difference of the transformations of the actual and predicted virtual measurements [34]

Finally, we define C as the set of pairs of indices i, j for which a virtual measurement exists. Based on the above, the goal of the smoothing SLAM approach is to find the *configuration of the node positions $x^\star$ that maximizes the log-likelihood of all observations in C* :

$$\mathbf{F}(x) = \sum_{\langle i,j \rangle \in C} F_{i,j} = \sum_{\langle i,j \rangle \in C} e_{i,j}^T \mathbf{\Omega}_{i,j} \, e_{i,j} \tag{2.1.25}$$

To express this as a least-squares problem, instead of maximizing, we can minimize the negative of the latter, thus:

$$x^\star = argmin_x \mathbf{F}(x) \tag{2.1.26}$$

Approximating the error function by its first order Taylor expansion we get the following:

$$e_{i,j}(\breve{x}_i + \Delta x_i, \breve{x}_j + \Delta x_j) = e_{i,j}(\breve{x} + \Delta x) \approx e_{i,j} + \boldsymbol{J}_{i,j}\Delta x \tag{2.1.27}$$

$$\begin{aligned}
\mathbf{F}_{i,j}(\breve{x} + \Delta x) &= e_{i,j}(\breve{x} + \Delta x)^T \boldsymbol{\Omega}_{i,j} e_{i,j}(\breve{x} + \Delta x) && (2.1.28)\\
&\approx (e_{i,j} + \boldsymbol{J}_{i,j}\Delta x)^T \boldsymbol{\Omega}_{i,j} (e_{i,j} + \boldsymbol{J}_{i,j}\Delta x) && (2.1.29)\\
&= \underbrace{e_{i,j}^T \boldsymbol{\Omega}_{i,j} e_{i,j}}_{c_{i,j}} + 2\underbrace{e_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{J}_{i,j}}_{b_{i,j}} \Delta x + \Delta x^T \underbrace{\boldsymbol{J}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{J}_{i,j}}_{\boldsymbol{H}_{i,j}} \Delta x && (2.1.30)\\
&= c_{i,j} + 2b_{i,j}\Delta x + \Delta x^T \boldsymbol{H}_{i,j}\Delta x && (2.1.31)
\end{aligned}$$

Using the latter expression in eqn. 2.1.31, and by setting:

$$\begin{aligned}
c &= \sum c_{i,j} && (2.1.32)\\
b &= \sum b_{i,j} && (2.1.33)\\
\boldsymbol{H} &= \sum \boldsymbol{H}_{i,j} && (2.1.34)
\end{aligned}$$

we rewrite eqn. 2.1.25 as follows:

$$\begin{aligned}
\mathbf{F}(\breve{x} + \Delta x) &= \sum_{\langle i,j\rangle \in C} F_{i,j}(\breve{x} + \Delta x) && (2.1.35)\\
&\approx \sum_{\langle i,j\rangle \in C} \Big[c_{i,j} + 2b_{i,j}\Delta x + \Delta x^T \boldsymbol{H}_{i,j}\Delta x \Big] && (2.1.36)\\
&= c + 2b^T\Delta x + \Delta x^T \boldsymbol{H}\Delta x && (2.1.37)
\end{aligned}$$

Given the quadratic form of eqn. 2.1.37 we now have to reach a formula suitable to solve with a non-linear system optimization framework like the Gauss-Newton. To do that we compute the partial derivative with regards to Δx and set it to 0:

$$\begin{aligned}
\frac{\partial \boldsymbol{F}(x + \Delta x)}{\partial \Delta x} &= \left(\boldsymbol{H} + \boldsymbol{H}^T\right)\Delta x + 2b \overset{\textbf{H symmetric}}{=} \\
&= 2\boldsymbol{H}\Delta x + 2b = 0 \Rightarrow \\
\Delta x^\star &= -\boldsymbol{H}^{-1} b && (2.1.38)
\end{aligned}$$

Up to now we have described a generic least-squares approach to reach eqn. 2.1.38[3]. However we haven't yet exploited the sparse and/or incremental nature of SLAM to facilitate the problem solution.

H is the information matrix of the system, since it is obtained by projecting the measurement error in the space of trajectories via the Jacobians. This matrix, (having not

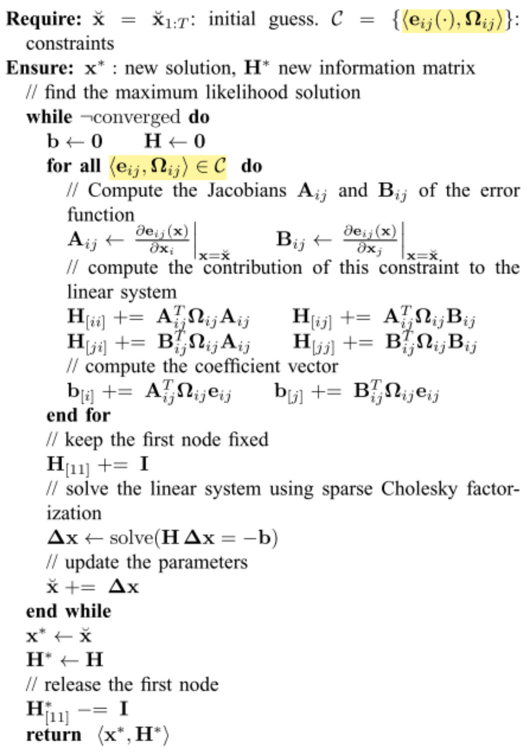
[3]For the mathematic formulation of the Levenberg-Marquardt algorithm for solving the derived least-squares non-linear problem in eqn. 2.1.38 see 3.1.3.

marginalized out previous robot poses) *is naturally sparse* as it contains non-zero blocks only between poses that are connected via a constraint. When the robot is actively exploring (traversing previously unknown parts of the environment) new entries are added to the end of the $\boldsymbol{H}$ matrix and b vector. Its sensors (e.g., laser scan) can associate its latest measurement/pose (which corresponds to its latest inserted rows and columns) with some of the poses that are relatively close to it. Thus, and assuming that there wasn't any loop closure detected (that is, no constraint between latest and first nodes), $\boldsymbol{H}$ is going to be a block diagonal matrix.

The linearised solution is obtained by adding the found differences to the initial estimation:

$$x^{\star} = \breve{x} + \boldsymbol{\Delta} x^{\star} \tag{2.1.39}$$

The algorithm is also illustrated in fig. 2.1.6



```
Require: $\breve{\mathbf{x}} = \breve{\mathbf{x}}_{1:T}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \mathbf{\Omega}_{ij} \rangle\}$:
  constraints
Ensure: $\mathbf{x}^*$ : new solution, $\mathbf{H}^*$ new information matrix
  // find the maximum likelihood solution
  while ¬converged do
    $\mathbf{b} \leftarrow \mathbf{0}$    $\mathbf{H} \leftarrow \mathbf{0}$
    for all $\langle \mathbf{e}_{ij}, \mathbf{\Omega}_{ij} \rangle \in \mathcal{C}$ do
      // Compute the Jacobians $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ of the error
      function
      $\mathbf{A}_{ij} \leftarrow \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$    $\mathbf{B}_{ij} \leftarrow \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$
      // compute the contribution of this constraint to the
      linear system
      $\mathbf{H}_{[ii]} \mathrel{+}= \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$    $\mathbf{H}_{[ij]} \mathrel{+}= \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$
      $\mathbf{H}_{[ji]} \mathrel{+}= \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$    $\mathbf{H}_{[jj]} \mathrel{+}= \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$
      // compute the coefficient vector
      $\mathbf{b}_{[i]} \mathrel{+}= \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$    $\mathbf{b}_{[j]} \mathrel{+}= \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$
    end for
    // keep the first node fixed
    $\mathbf{H}_{[11]} \mathrel{+}= \mathbf{I}$
    // solve the linear system using sparse Cholesky factor-
    ization
    $\mathbf{\Delta x} \leftarrow \text{solve}(\mathbf{H}\,\mathbf{\Delta x} = -\mathbf{b})$
    // update the parameters
    $\breve{\mathbf{x}} \mathrel{+}= \mathbf{\Delta x}$
  end while
  $\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$
  $\mathbf{H}^* \leftarrow \mathbf{H}$
  // release the first node
  $\mathbf{H}^*_{[11]} \mathrel{-}= \mathbf{I}$
  return $\langle \mathbf{x}^*, \mathbf{H}^* \rangle$
```

Figure 2.1.6: Pseudocode for computing the mean $\boldsymbol{x}^{\star}$ and information matrix $\boldsymbol{H}^{\star}$ of the multivariate Gaussian approximation of the robot pose posterior from a graph of constraints[34]

Dimensions of relevant terms

According to equations 2.1.33, 2.1.34 the matrix $\boldsymbol{H}$ and the vector $\boldsymbol{b}$ are computed by summing up a set of matrices and vectors, one for each constraint, that is each constraint contributes to the system with an addend term. The structure of the latter depends only on the Jacobian of the error function. Since the error function of a constraint depends only on the values of two nodes, it has the following form:

$$\boldsymbol{J}_{i,j} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & \underbrace{\boldsymbol{A}_{i,j}}_{\text{node}\, i} & \mathbf{0} \cdots & \mathbf{0} & \underbrace{\boldsymbol{B}_{i,j}}_{\text{node}\, j} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \tag{2.1.40}$$

where $\boldsymbol{A}_{i,j}, \boldsymbol{B}_{i,j}$ are derivatives of the error function with regards to $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

We also have the following forms for $\boldsymbol{H}_{i,j}$ and $\boldsymbol{b}_{i,j}$

$$\boldsymbol{H}_{i,j} = \begin{bmatrix} \ddots & & & & \\ & \boldsymbol{A}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{A}_{i,j} & \cdots & \boldsymbol{A}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{B}_{i,j} & \\ & \vdots & \ddots & \vdots & \\ & \boldsymbol{B}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{A}_{i,j} & \cdots & \boldsymbol{B}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{B}_{i,j} & \\ & & & & \ddots \end{bmatrix} \tag{2.1.41}$$

$$\boldsymbol{b}_{i,j} = \begin{bmatrix} \vdots \\ \boldsymbol{A}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{e}_{i,j} \\ \vdots \\ \boldsymbol{B}_{i,j}^T \boldsymbol{\Omega}_{i,j} \boldsymbol{e}_{i,j} \\ \vdots \end{bmatrix} \tag{2.1.42}$$

2.2 Multi-Robot SLAM

The current section is based on the excellent multiple robot review presented in [3]. Its goal is to outline important challenges, problems and limitations that arise when extending SLAM from a single agent to a team of robots.

2.2.1 Data manipulation and flow in multi-robot SLAM

Data Communication

The capability of a system to share data among robots is a key requirement in multiple-robot SLAM. Information between robots can be exchanged via communication channels that might not be available at all times, in all places. Bandwidth and coverage of the communication network are two important factors for the performance of SLAM.

Data Sharing

Sharing data among robots is a fundamental issue in multiple-robot SLAM. Past approaches to collaborative SLAM can generally be categorized based on whether they share raw sensor data [43] or processed data [44]. Raw sensor data means that sensed information, such as laser ranger measurements and wheel odometry readings, are not processed. In processed data, such as a map or poses of robots, sensor readings are processed through filtering, smoothing or other methods. Sharing raw sensor data results in more flexibility but requires higher bandwidth, stable communication between the robots as well as more processing power. On the contrary, sharing maps uses less bandwidth and there is no need to process raw data redundantly; however, the performance is dependent on the quality of maps.

Data Distribution

Distribution of data can be further divided into the following categories:

Centralized
In a centralized system, the computation for a task is performed by a predefined robot of the team or by an external agent. The central agent processes the incoming data and provides the required information and feedback to other agents.

Decentralized
In a decentralized system, the computation for a task is performed by multiple robots of the team. Obviously, this structure requires that the robots have enough computational power to respond to the processing demands.

Distributed
In a distributed system, the computation for a task is divided among the robots of the team.

Notice that these definitions depend on the corresponding task. As an example, the computational model for the production of a global map of the environment may be implemented in a central node, but the SLAM execution may also be distributed, meaning that each agent does its own mapping and communicates accordingly with its neighbors.

Data Processing
The choice of the strategy for estimating the poses of robots and the global map depends on many factors, such as the available memory, the processing capability and the type of the sensor data. The available SLAM implementations, such as EKF, Particle Filtering or graphSLAM, have different demands and computational complexity. The same holds for the corresponding extensions of each of the aforementioned methods in multi-robot SLAM.

2.2.2 Problems in multi-robot SLAM

In this section, problems for multi-robot SLAM are listed and explained briefly. The list includes 10 major problems, along with a summary for each one [3]. Due to the nature of the selected approach in this thesis, extra attention is payed to graph-based strategies.

Relative Poses of Robots
One of the most critical issues in multi-robot SLAM is for a robot to decide what is the relative transformation of its map/trajectory to that of its counterpart. In multiple-robot SLAM, the map provided by each robot in its own reference coordinates is called the local map. Each local map is generated from coordinated measurements such as laser scans. Each robot tries to integrate all of the local maps provided by the other robots to generate a global map of the environment. However, this is a difficult task, because the required alignments or transformation matrices, which relate these maps to each other, are unknown in general. The problem of the relative pose of the robot is coupled with the multiple-robot data association problem. Knowledge of the one renders the other as a simple problem.

Uncertainty of the Relative Poses
Even in cases that one robot successfully finds a valid transformation from its own origin to the origin of a neighboring robot, it hasn't yet placed this transformation in a probabilistic context, meaning the uncertainty of this transformation is not considered. An example of this is the rendezvous technique used in [43] in which the inter-robot transformation at rendezvous is considered certain and using that, each

robot ``plays back'' the measurements received by its neighbors. However, not considering that uncertainty introduces errors during the playback of the received data.

Updating Maps and Poses

Once the relative transformation is found, a procedure is required to fuse local maps. The resulting map should integrate all information from given local maps. As a result of updating the maps, poses of the robots should also be updated. This requires considering the trajectory of the robots and new information received from other maps. Because of the nature of multiple-robot SLAM, updating poses and maps is a coupled problem. In feature-based SLAM, data association and finding the correspondence between duplicate landmarks across the robots is an important part of updating maps as well.

Complexity

Robotics applications are usually real time. Thus, it is very important to design an algorithm capable of solving the previously mentioned problems with minimum time and memory requirements. In multiple-robot SLAM,space complexity and time complexity are two important issues and both affect the scalability of the operation.

Communications

Communication is a key issue in multi-robot cooperation and as well as coordination tasks. Its significance increases when operating in networks with bandwidth restrictions, limited communication range of each robot or even worse in situations that no prior network infrastructure exists. While popular approaches in multi-robot SLAM assume no restriction in the amount/frequency of data exchange [43], [32] certain works in the field [45], [6], [46] impose realistic constraints. As an example, authors of [6] propose the exchange of condensed measurements as a way of significantly reducing the data sent over the network ans also provide a thorough analysis of the amount of data actually transferred over the wire.

Line-of-sight Observations

On some occasions, robots can see and detect one another. This may be implemented using visual sensors and corresponding markers (e.g., monocular cameras and Aruco markers). Each robot might already have an estimate of the pose of other robots. However, when robots can see one another through line-of-sight and direct observations, these estimates can be improved. This fact can help robots to reduce mapping and localization error. In most applications and especially in close-range localization, line-of-sight observations are much more reliable than other indirect estimation techniques.

Heterogeneity in Vehicles and Sensors

An important advantage of team-based mapping is that different types of robots, equipped with different sensors, can provide a better model of the environment. As an example, authors of [32] employed square-root information smoothing via the iSAM module in a real-time experiment with a mobile and a quadcopter. Each robot exploited different parts of the environment, that their counterpart could not have traversed. Another example is the work in [47] in which a team of heterogeneous robots, including a quadrotor and two ground robots, map a multi-floor earthquake-damaged building collaboratively.

Synchronization

As a general rule, each acquired sensor reading should have a time stamp field, which shows the time of the acquisition of the data. An important issue in a system of multiple agents and multiple sensors is the synchronization of the clocks. Synchronization can be considered at two levels: first, local synchronization,which means the sensors of each robot should be synchronized and second, global synchronization, which means all robots on the team must have synchronized clocks. To synchronize time on different robots the Chrony software[48], also utilised in ROS middleware, is a suitable choice: Chrony supports online and offline time adjustment by two different applications. In the online case, a network time protocol (NTP) daemon runs in the background and synchronizes the time with time servers. For an isolated machine, one might enter the time periodically. The synchronized time appears as a label in the header of each acquired data

Performance measure

In multiple-robot SLAM, evaluating the accuracy of results is a challenging problem due to the lack of the model of the environment and the actual trajectory of the robots. Additionally, evaluating the accuracy of SLAM becomes more critical when the robots rely on the SLAM to perform autonomous behaviors. Therefore, performance measure is always required to determine the reliability of multiple-robot SLAM.

2.2.3 Solutions and notable works

The following is list of papers that made an impact in the field of multi-robot SLAM. These are sorted in ascending order:

- In [49] authors extend their single robot algorithm, the ``Constrained Local Submap Filter'' [50], [51] in the context of multiple robots. Overall, it can be characterized as a centralized landmark-based SLAM approach that can handle and update a global map using local submaps from multiple agents. The latter is feasible since a transformation from each robot to the global frame of reference is considered known. Given the categorization of multi-robot SLAM problems the strategy at hand deals with the following:
  - Complexity
  - Update of maps and poses
- In [52] due to the additivity property of the information form, information-based SLAM is more suitable in the multi-robot context than its covariance-based counterpart. As Nettleton et al. noticed in [53], updating maps is possible in time logarithmic in the number of the robots in the team. Adding to this, Thrun and Liu implement a multi-robot SLAM scheme based on Sparse Information Filters (SEIF). In their research they deal with two main problems, notably:
  - Finding the relative poses
  - Updating maps and poses

  The former is possible by matching landmark features in the robot maps. Finding landmark correspondences is also accelerated by storing them in a kd-tree form. Once a rough estimate is calculated, it can be optimized by minimizing the quadratic displacement of the features. Furthermore, handling the SLAM problem in information form enables straightforward fusion of duplicate landmarks as well. Assume that we have the following information matrix and vector of a map with four features as follows:

$$\begin{bmatrix} \Omega_{1,1} & \Omega_{1,2} & \Omega_{1,3} & \Omega_{1,4} \\ \Omega_{2,1} & \Omega_{2,2} & \Omega_{2,3} & \Omega_{2,4} \\ \Omega_{3,1} & \Omega_{3,2} & \Omega_{3,3} & \Omega_{3,4} \\ \Omega_{4,1} & \Omega_{4,2} & \Omega_{4,3} & \Omega_{4,4} \end{bmatrix}, \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix} \tag{2.2.1}$$

As an example, if features 2 and 4 are duplicates, the corresponding matrix and vector after fusion will be as follows:

$$\begin{bmatrix} \Omega_{1,1} & \Omega_{1,2} + \Omega_{1,4} & \Omega_{1,3} \\ \Omega_{2,1} + \Omega_{4,1} & \Omega_{2,2} + \Omega_{4,2} + \Omega_{2,4} + \Omega_{4,4} & \Omega_{2,3} + \Omega_{4,3} \\ \Omega_{3,1} & \Omega_{3,2} + \Omega_{3,4} & \Omega_{3,3} \end{bmatrix}, \begin{bmatrix} \zeta_1 \\ \zeta_2 + \zeta_4 \\ \zeta_3 \end{bmatrix}^{4} \tag{2.2.2}$$

- Zhou and Roumeliotis [54] are the first to utilize robot rendezvous (either random ones, or prearranged) in the SLAM problem. Their approach comprises a feature-based SLAM that uses EKF for filtering robot poses and landmark positions. In robot rendezvous, four main problems of sec. 2.2.2 are addressed:
  - Unknown relative poses
  - Uncertainty of the relative poses
  - Update of maps and poses
  - Complexity

  Notice that by imposing a rendezvous prior to utilising the data collected by the neighbor robot, authors bypass the potentially costly computation of relative transformation, and also ensure that an overlap of the robot maps and trajectories actually exists. Authors also employ the *Sequential Nearest Neighbor Test* for solving the data-association and decide on potential duplicate landmarks.

- Howard in [43] proposes a particle filtering method using a view-based representation of the map. His approach solves the following problems:
  - Relative poses of robots
  - Update of maps and poses
  - Closing loops

  Robots without knowing their relative position with regards to their neighbors start executing independent PF-SLAM. When robots meet, they compute an inter-robot transformation of their current poses using onboard cameras and markers. This way, and having exchanged all observations gathered so far, each robot plays back the received measurements starting from the rendezvous point until the other robot's initial position. However this approach utilizes only the first rendezvous of two robot agents while it doesn't consider the uncertainty of the computed transformation. These points are dealt with in [45] in which the authors also relax the unlimited bandwidth assumption of Howard's paper.

- One of the most prominent graphSLAM multi-robot implementations was proposed by Kim et al., in [32], via the extension of the iSAM algorithm to heterogeneous multi-robot SLAM. Robot trajectories are handled as independent pose

[4]Note that the mean and covariance of the EKF form are transformed into an information matrix $\boldsymbol{\Omega}$ and an information vector Z using the equations $\Omega_t = \Sigma_t^{-1}$, $\zeta = \mu^T \Sigma_t^{-1}$

graphs among which, connections are formed (inter-graph constraints) when there is a direct or indirect robot encounter.[5] Kim et al., also introduce the notion of anchor nodes, which are used to specify the offset of each trajectory with respect to a common global frame. After an encounter of two robot agents, their pose-graphs can be optimized as a whole, since there are adequate constraints between them.

[5]We define a ``direct encounter'' as the case where one robot directly sees its counterpart, (e.g., via the root onboard sensors and fiducial markers) whereas an indirect encounter is the measurement of an area previously scanned by another agent.

Chapter 3

Single-robot SLAM development

Current section makes a thorough analysis of the single-robot graphSLAM algorithm that was developed. More specifically, we analyze the work done for the *MRPT* organization during the *Google Summer of Code 2016* program (sec. 3.2). Details into the wrapper codes for using the developed MRPT library in the Robot Operating System (ROS) (sec. 3.5), as well as experimental results (sec. 3.6) are also provided.

3.1 Mathematical Background

Current section includes mathematical details on algorithms that are utilized in the single-robot graphSLAM algorithms (sec. 3) as well as the multi-robot ones (sec. 5).

Overall the following are presented:

Iterative Closest Point - (sec. 3.1.1)
Algorithm used to associate two 2D/3D range scans and decide on a transformation that maximally aligns one scan to the other.

Robust Loop Closure scheme - (sec. 3.1.2)
Association scheme that determines if a set of potential hypotheses are indeed valid loop closure edges

Levenberg-Marquardt non-linear solver - (sec. 3.1.3)
Non-linear system solver used as the optimization module of choice in single and multi-robot SLAM algorithms that have been developed during the thesis.

3.1.1 Iterative Closest Point (ICP) Algorithm

The Iterative Closest Point, or Iterative Corresponding Point (ICP) is an algorithm used to find a valid transformation with which one point cloud (source) is maximally aligned to another (reference). ICP was first introduced in [55].

ICP is widely used for registering the outputs of 2D/3D scanners. ICP starts with two point clouds / meshes as well as an initial guess for their relative rigid-body transform, and iteratively refines the transform by repeatedly generating pairs of corresponding points on the meshes and minimizing an error metric [56].

In our implementation, ICP is used to add additional edges/constraints between already registered nodes in the graph by associating (finding a valid transform) between the laser scans of those nodes. By doing so, it introduces additional , most-likely inter-conflicting, constraints in the optimization stage of graphSLAM, thus after optimization, improving the final node positions. This is a very important step in the overall algorithm execution, since odometry measurements can only introduce a constraint between consecutive node positions.

Algorithm stages

Although many variants/modifications to the classic ICP have been introduced, we can classify them according to the following stages [56]:

1. **Selection** of some set of points in one or both meshes.
2. **Matching** these points to samples in the other mesh.
3. **Weighting** the corresponding pairs appropriately.
4. **Rejecting** certain pairs by looking at each pair individually or by considering an entire set of pairs.
5. Assigning an **error metric** based on the point pairs.
6. **Minimizing** the error metric.

After minimizing the error metric and acquiring the corresponding transformation, the target reference is transformed by the computed transform and ICP continues to the next iteration.

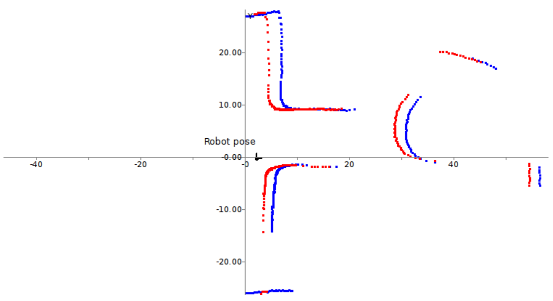
For the MRPT implementation of the ICP algorithm (defined in the `mrpt::slam::CICP` class) the following remarks can be made:

- **Input:** Two 2D/3D maps, (optional) initial estimation of the transform from one to another.

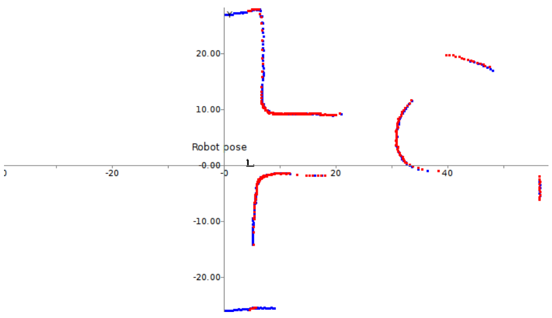
  Output: Probability density function (PDF) of the output transform for aligning the given maps.
- In the case of 2D point maps (applicable in our implementation) and to accelerate the nearest neighbor search, a KD-treee is also employed.
- Selection stage essentially comes down to the decimation of the given laser scans by a factor given by the user of the class (by default `decimation = 5`).
- The error metric minimization is implemented as a least squares minimization between the matched points of the two decimated lists. [1]
- A progressive refinement of the ICP estimation is also implemented using the α parameter (read Iterative Closest Point (ICP) and other matching algorithms in MRPT for more information)

An example of using the ICP algorithm of MRPT to find the alignment transformation is also visualized for a set of scans in fig. 3.1.1.

[1]For the mathematical part on finding the ideal rigid body transformation from a set of matched points, see [57]



(a) 2D point maps prior to the ICP operation



(b) 2D point maps after the ICP operation. The suggested transformation after 32 iterations is $[4.278, -0.034, 0.361°]$.Elapsed time:$1.399\mu s$

Figure 3.1.1: Example of using the ICP algorithm of MRPT to align two point maps derived from their respective laser scans

3.1.2 Robust Loop-Closure scheme

In the current section, we present the mathematical formulas that have been used for implementing the robust hypotheses evaluation scheme in loop closure situations. Analysis is heavily based on [40].

Purpose and usage advantages

We chose to implement Olson's robust loop closure scheme in the single-robot SLAM context as it offered the following advantages:

- Does not require landmark covariances as the majority of data-association algorithms do, so it can be combined with algorithms that don't explicitly store landmark covariances(e.g., FastSLAM, information filters).
- Computes the second best set of mutually consistent hypothesis which is *orthogonal* to the first. So if both the 1^{st} and 2^{nd} hypotheses can interpret the data equally well, we reject the hypothesis due to potential picket-fence problem.
- Can be executed in real-time and is adaptable to sensor modalities.
- It is a popular approach in the robotics community and is used in a variety of SLAM applications.

Theory and Implementation details

Having parsed the sensor measurements and having constructed an initial graph from the odometry constraints and from scan-matching of neighboring nodes, we need to identify *when* and *how* to determine loop closing incidents. This is implemented based on [40]. We present a figure of the approach in fig. 3.1.2 and we summarize the steps below:

- Instead of using Olson's proposed grouping method, we used Jose Luis Blanco's hybrid metric-topological grouping approach[58] [2].
- Having assembled the groups of nodes, we find the groups that might contain loop closure edges (these groups contain successive nodes with large difference in their

[2]Readers are encouraged to consult [58] for design and implementation details.

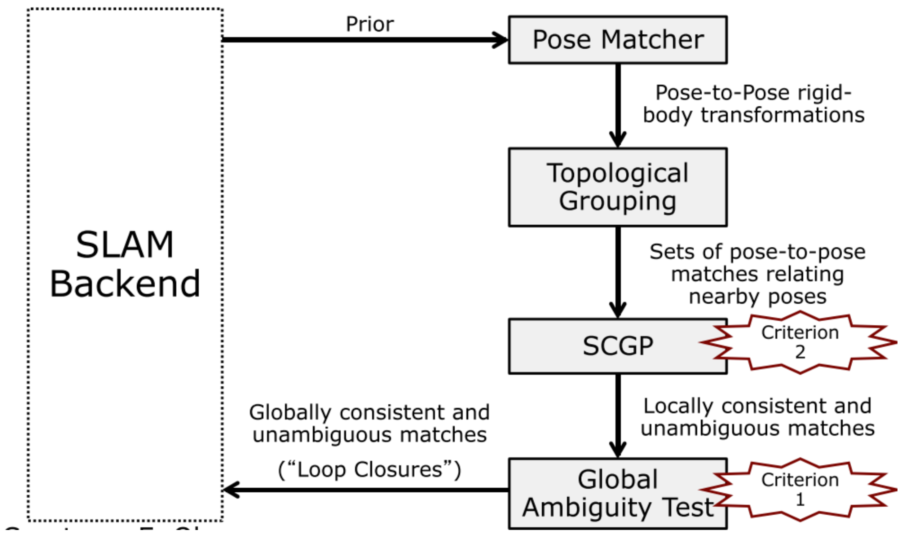
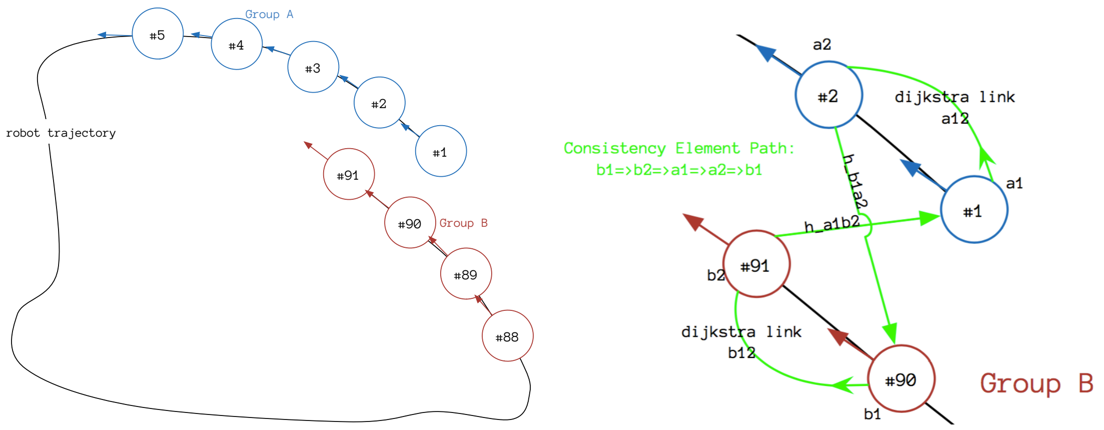

SLAM Backend
Prior
Pose Matcher
Pose-to-Pose rigid-body transformations
Topological Grouping
Sets of pose-to-pose matches relating nearby poses
SCGP
Criterion 2
Locally consistent and unambiguous matches
Global Ambiguity Test
Criterion 1
Globally consistent and unambiguous matches ("Loop Closures")

Figure 3.1.2: Suggested data association scheme



(a) Typical loop closure situation. Robot has returned to its origin and acquires observations of areas that it has already visited.

(b) Formation of one hypotheses group, containing hypotheses to be evaluated and dijkstra links between nodes of the same node groups

Figure 3.1.3: Sketches of a loop closure situation

IDs). These groups are then split into two subgroups A, B, with the former containing the lower node IDs and the latter the higher. To minimize computational cost as well as the possibility of wrong loop closure registration, we search for loop closures only between the last nodes of group B and the first nodes of group A (see fig. 3.1.3(a) for a schematic representation of the loop closure situation).

- Having identified the potential hypotheses in each graph subgroup, we test them for local consistency. Based on[40] the potential loop closure edges are not evaluated individually but rather *in sets*. Hypotheses are grouped together spatially and *hypothesis sets* are formed between *dijkstra odometry links* and *candidate hypotheses links*. By combining two candidate hypotheses with two odometry constraints we can form a loop. If the aforementioned hypotheses are valid, then the composition of the rigid body transformations around the loop should equal the identity matrix. The pairwise consistency of hypotheses i, j is given by eqn. 3.1.1 and eqn. 3.1.2.

$$A_{i,j} = e^{-T\Sigma_T T^\intercal}, \tag{3.1.1}$$

where T is the composition of transforms of the two dijkstra links and the hypotheses i, j.

All the individual $A_{i,j}$ can be merged into a consistency matrix A. The validation of the hypothesis now comes down into finding a subset of hypotheses whose pairwise consistency is, on average, the greatest [40]. Let u be an indicator vector such that $u_i = 1$ if the i^{th} hypothesis is to be accepted, and $u_i = 0$ otherwise. The average pairwise consistency $\lambda(u)$ for a subset of hypotheses u is: [3].

$$\lambda(u) = \frac{u^\intercal A u}{u^\intercal u} \tag{3.1.2}$$

Now making the assumption that u is a continuous variable, we can find the look for extrema of $\lambda(u)$ by differentiating eqn. 3.1.2 and setting the derivative to 0 with respect to u. Doing that we end up with the following eigenvalue-eigenvector problem:

$$Au = \lambda(u)$$

Eqn. 3.1.2 is solved numerically using power iterations, and the first two eigenvalues (orthogonal with each other) are computed.

Finally we convert u back to the discrete domain using a predetermined threshold t. Let $w_i(t)$ be the discretized version

$$w_i(t) = \begin{cases} 1, & \text{if } u_i \geq t \\ 0, & \text{otherwise} \end{cases} \tag{3.1.3}$$

The confidence of the hypothesis set can be measured by computing the ratio of the largest two eigenvalues, $\frac{\lambda_1}{\lambda_2}$. When this ratio is larger than a threshold, then we are confident about the hypotheses set and it can be accepted. Otherwise we are in risk of picket-fence problem and the hypotheses set is rejected.

- The potential hypotheses is finally tested for global consistency [4]. Having con-

[3]for details on the derivation see [40]

[4]Step is not yet implemented in the CLoopCloserERD class

structed two local maps of its environment (e.g., A, B) at different points along its trajectory, we should now determine whether the A, B represent the same location, or whether they are two physically distinct places [40]. We begin by assuming that we have some prior knowledge of the relative position of the two environments (as in SLAM). This allows us to compute a bounding ellipse relative to area B in which area A must be found. Naturally, if B is not contained within this ellipse, B cannot be the same location as A. The more interesting case is when local map B is within the ellipse that contains local map A. The main idea is that a local match is globally consistent if two conditions are satisfied. The first condition requires that the local match cover a spatial area that is comparable in size to the uncertainty ellipse. If the uncertainty ellipse is large enough to contain two or more identical-looking regions that might match area A, then we cannot be sure that area A is the same as area B. Conversely, if the uncertainty ellipse is only large enough to hold one "copy" of the locally matched region, then A and B must be the same location, since two distinct regions of that size could not both exist within the uncertainty ellipse.

Inconsistencies in Olson's paper

Olson uses the following formula for evaluating the pairwise consistency between two hypotheses i,j:

$$A_{i,j} = e^{T\Sigma_T^{-1}T^\intercal} \tag{3.1.4}$$

Where:

- T is the rigid body transformation using the two hypotheses and the two Dijkstra Links connecting the nodes that form the hypotheses
- Σ_T is the covariance matrix of the aforementioned rigid body transformation

However this formula is inconsistent with the rest of the paper explanations and mathematical formulas:

- The author states that:

  This quantity is proportional to the probability density that the rigid-body transformation is the identity matrix, thus transformation matrix is a $0_{3\times1}$ vector (i.e., $T = [0, 0, 0]^T$).

  This is inconsistent with the given formula. Suppose that a wrong loop closure is included in the $A_{i,j}$, therefore the pairwise-consistency element should have a low value. For this to hold true, the exponent of the consistency element should be small and, neglecting the covariance matrix of rigid-body transformation (e.g., unit covariance matrix), $TT^\intercal$ should be small. When a wrong loop closure is evaluated the aforementioned quantity increases since the hypotheses do not form a correct loop. Therefore the worse the rigid-body transformation the higher the exponent term, therefore the higher the consistency element.

- Author uses the information matrix Σ_T^{-1} in the exponential. However in the optimal case (high certainty of two correct loop closure hypotheses) information matrix and rigid body transformation vector T have opposite effects in the exponent term:

  - Correct loop closure $\Rightarrow T \rightarrow [0, 0, 0] \Rightarrow$ exponent $\downarrow$
  - Correct loop closure $\Rightarrow$ diagonal_terms(Σ_T^{-1}) $\uparrow\Rightarrow$ exponent $\uparrow$

3.1.3 Efficient Least-Squares problem solving - Levenberg-Marquardt Algorithm

Assume a vector function $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ with $m \geq n$. Goal of non-linear system solvers is to try and minimize $\left\|\mathbf{f}(\mathbf{x})\right\|$, or equivalently find:

$$x^{\star} = argmin_x\{F(\mathbf{x})\}, \tag{3.1.5}$$

where

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}\Big(f_i(\mathbf{x})\Big)^2 = \frac{1}{2}\left\|\mathbf{f}(\mathbf{x})\right\|^2 = \frac{1}{2}\mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x}),\ F : \mathbb{R}^n \mapsto \mathbb{R} \tag{3.1.6}$$

Notice the $\frac{1}{2}$ term in front of the sum in eqn. 3.1.6. This is used to avoid terms of 2 that arise in later partial differentiations of this formula. Provided that **f** has continuous second partial derivatives, we can write its *Taylor expansion* as follows:

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) = \mathbf{f}(\mathbf{x}) + \mathcal{J}(\mathbf{x})\mathbf{h} + \mathcal{O}(\|\mathbf{h}\|^2) \tag{3.1.7}$$

where $\mathcal{J} \in \mathbb{R}^{m \times x}$ is the Jacobian matrix:

$$\mathcal{J}(\mathbf{x})_{i,j} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}) \tag{3.1.8}$$

Using eqn. 3.1.6 we also have:

$$\frac{\partial F}{\partial x_j}(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})\frac{\partial f_i}{\partial x_j}(\mathbf{x}). \tag{3.1.9}$$

Thus, the gradient can be written as:

$$\mathbf{g} \equiv \mathbf{F}'(\mathbf{x}) = \begin{pmatrix} \frac{\partial F}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial F}{\partial x_n}(\mathbf{x}) \end{pmatrix} \tag{3.1.10}$$

In the following paragraphs we provide an analysis of the Levenberg-Marquardt non-linear solver which we use extensively in both the single and multi-robot graphSLAM implementations, in the rest of the thesis. Analysis is based on the work in [2]. We also provide an overview of the Levenberg-Marquardt algorithm in fig. 3.1.4.

```
begin
  $k := 0; \quad \nu := 2; \quad \mathbf{x} := \mathbf{x}_0$
  $\mathbf{A} := \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}); \quad \mathbf{g} := \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
  $found := (\|\mathbf{g}\|_\infty \leq \varepsilon_1); \quad \mu := \tau * \max\{a_{ii}\}$
  while (not found) and ($k < k_{\max}$)
    $k := k+1; \quad$ Solve $(\mathbf{A} + \mu\mathbf{I})\mathbf{h}_{\text{lm}} = -\mathbf{g}$
    if $\|\mathbf{h}_{\text{lm}}\| \leq \varepsilon_2(\|\mathbf{x}\| + \varepsilon_2)$
      $found :=$ true
    else
      $\mathbf{x}_{\text{new}} := \mathbf{x} + \mathbf{h}_{\text{lm}}$
      $\varrho := (F(\mathbf{x}) - F(\mathbf{x}_{\text{new}}))/(L(\mathbf{0}) - L(\mathbf{h}_{\text{lm}}))$
      if $\varrho > 0$                                {step acceptable}
        $\mathbf{x} := \mathbf{x}_{\text{new}}$
        $\mathbf{A} := \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}); \quad \mathbf{g} := \mathbf{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x})$
        $found := (\|\mathbf{g}\|_\infty \leq \varepsilon_1)$
        $\mu := \mu * \max\{\frac{1}{3}, 1 - (2\varrho - 1)^3\}; \quad \nu := 2$
      else
        $\mu := \mu * \nu; \quad \nu := 2 * \nu$
end
```

Figure 3.1.4: Levenberg-Marquardt pseudocode [2]

Originally published in 1944 by Kenneth Levenberg [59] and later rediscovered in 1963 by Donald Marquardt [60], the Levenberg-Marquardt algorithm (LMA) also known as damped least-squares (DLS) method is used to solve non-linear least squares problems. LMA an iterative technique that locates the minimum of a multivariate function that is expressed as the *sum of squares of non-linear real-valued functions* [61].

It comprises a combination of the Gauss-Newton algorithm (GNA) and the steepest descent. More specifically, the formula that computes the step towards the solution (i.e., $\mathbf{h}_{lm}$) is provided by the following modification to the standard Gauss-Newton formula:

$$\left(\boldsymbol{\mathcal{J}}^T\boldsymbol{\mathcal{J}} + \mu\mathbf{I}\right)\mathbf{h_{lm}} = -\mathbf{g} \tag{3.1.11}$$

$$\text{where the gradient is: } \mathbf{g} = \boldsymbol{\mathcal{J}}^T\mathbf{f}$$

$$\mu \geq 0$$

In eqn. 3.1.11 $\boldsymbol{\mathcal{J}}$ and $\mathbf{f}$ are multivariate functions of the state $\mathbf{x}$. Given the same equation, we notice the following effect of the μ damping coefficient:

1. For all $\mu > 0$ the coefficient matrix is *positive definite* and that ensures that $\mathbf{h_{lm}}$ is a descent direction
2. In case μ also has a large value:

   $$\mathbf{h}_{lm} \simeq -\frac{1}{\mu}\mathbf{g} = -\frac{1}{\mu}\mathbf{F}'(\mathbf{x}),$$

   that is, a short step in the steepest descent direction.
3. If μ is very small ($\mu \to 0$), then $\mathbf{h}_{lm} \simeq \mathbf{h}_{gn}$. This is the strategy to use in the late stages of the algorithm when the estimated state is close to the solution (i.e., $x^\star$). Additionally if $\mathbf{F}(\mathbf{x}^\star) \propto 0$ then the convergence speed is quadratic due to the nature of GNA.

4. From the previous 2 points, we conclude that the damping term affects both the *direction* as well as the *size* of the step.

The choice of initial μ-value should be related to the size of the elements in the initial estimation of $\mathbf{A_0} = \boldsymbol{\mathcal{J}}^T \boldsymbol{\mathcal{J}}$ for $\mathbf{x} = \mathbf{x_0}$. This is done by the following equation:

$$\mu_0 = \tau max_i \alpha_{ii}^{(0)} \tag{3.1.12}$$

where τ is a user-defined variable[5]

We can further define the *gain ratio*. This quantity evaluates the quality of the current linearised model and essentially constitutes a ratio of the actual to the predicted decrease in function value, that is:

$$\rho = \frac{F(\mathbf{x}) - F(\mathbf{x} + \mathbf{h}_{lm})}{L(\mathbf{0}) - L(\mathbf{h}_{lm})}, \tag{3.1.13}$$

where the denominator is the gain predicted by the linearised model of F. The latter is derived from 1^{st} order Taylor expansion of $\mathbf{f}$, that is:

$$\begin{aligned}
\mathbf{f}(\mathbf{x} + \mathbf{h}) &\simeq \ell(\mathbf{h}) \equiv \mathbf{f}(\mathbf{x}) + \boldsymbol{\mathcal{J}}(\boldsymbol{x})\mathbf{h}, & (3.1.14)\\
F(\mathbf{x} + \mathbf{h}) &\simeq L(\mathbf{h}) \equiv \frac{1}{2}\ell(\mathbf{h})^T \ell(\mathbf{h}) \\
&= \frac{1}{2}\mathbf{f}^T\mathbf{f} + \mathbf{h}^T J^T f + \frac{1}{2}\mathbf{h}^T \boldsymbol{\mathcal{J}}^T \boldsymbol{\mathcal{J}} \mathbf{h} \\
&= F(\mathbf{x}) + \mathbf{h}^T \boldsymbol{\mathcal{J}}^T \mathbf{f} + \frac{1}{2}\mathbf{h}^T \boldsymbol{\mathcal{J}}^T \boldsymbol{\mathcal{J}} \mathbf{h} & (3.1.15)
\end{aligned}$$

We can now compute the denominator of eqn. 3.1.13 as follows:

$$\begin{aligned}
L(\mathbf{0}) - L(\mathbf{h}_{lm}) &= -\mathbf{h}_{lm}^T \boldsymbol{\mathcal{J}}^T \mathbf{f} - \frac{1}{2}\mathbf{h}_{lm}^T \boldsymbol{\mathcal{J}}^T \boldsymbol{\mathcal{J}} \mathbf{h}_{lm} \\
&= -\frac{1}{2}\mathbf{h}_{lm}^T \left(2\mathbf{g} + (\boldsymbol{\mathcal{J}}^T \boldsymbol{\mathcal{J}} + \mu \mathbf{I} - \mu \mathbf{I})\mathbf{h}_{lm}\right) \\
&= \frac{1}{2}\mathbf{h}_{lm}^T (\mu \mathbf{h}_{lm} - \mathbf{g}) & (3.1.16)
\end{aligned}$$

Note that since both $\mathbf{h}_{lm}^T \mathbf{h}_{lm}$ and $-\mathbf{h}_{lm}^T \mathbf{g}$ are guaranteed to be positive, $L(\mathbf{0}) - L(\mathbf{h}_{lm})$ is also guaranteed to be a positive quantity.

Furthermore, we can differentiate based on the computed value of ρ as follows:

- A large value of ρ indicates that $L(\mathbf{h}_{lm})$ is a good approximation to $F(\mathbf{x} + \mathbf{h}_{lm})$ and thus, we can decrease μ so that the next LMA step is closer to the corresponding GNA. This is done so that we utilize the GNA quadratic speed close to the actual solution.

- If ρ is small, maybe even negative, then $L(\mathbf{h}_{lm})$ is a poor approximation, and we should increase μ with the twofold aim of getting closer to the steepest descent direction **and** reducing the step length.

[5]LMA is not very sensitive to the choice of τ but as a rule of thumb, if x_0 is a good approximation to $x^\star$, this should be a small value (e.g., $\tau = 10^{-6}$) otherwise set it to 10^{-3} or even 1.

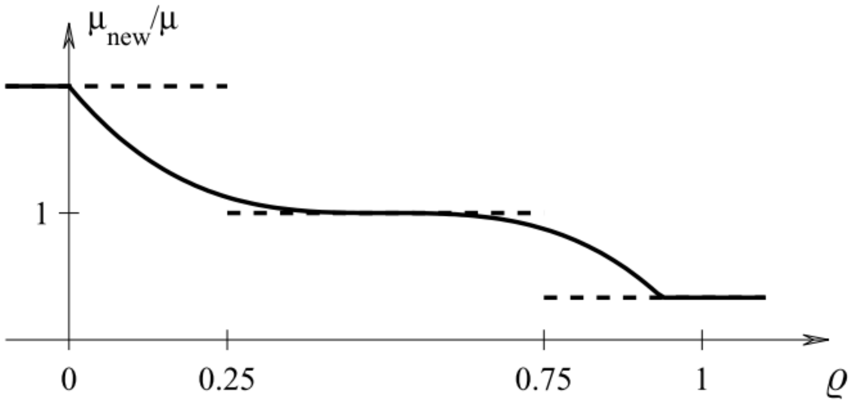


Figure 3.1.5: Full line: Updating μ - smooth version with $\nu = 2$. Dashed line: Initial suggested by Marquardt. [2]

Thus, based on the value of ρ one could modify μ to change the convergence procedure. For example the following criterion, initially proposed by Marquardt, could be used:

Algorithm 1 Example algorithm for modifying μ term - Discrete values

```
1 if $\rho < 0.25$ then
2     $\mu \leftarrow \mu \times 2$
3 else if $\rho \geq 0.75$ then
4     $\mu \leftarrow \mu/3$
5 end if
```

However this method is not sensitive to minor changes in the thresholds 0.25 and 0.75. Experience in using this criterion, shows that the discontinuous changes across the thresholds 0.25 and 0.75 can give rise to a *flutter* that slows down convergence. The criterion in the next algorithm solves this problem. Comparison of the two criteria can be illustrated in fig. 3.1.4.

Algorithm 2 Example algorithm for modifying μ term - Smooth version

```
1 if $\rho > 0$ then
2     $\mu \leftarrow \mu \times max\{\frac{1}{3}, 1 - (2\rho - 1)^3\}$
3     $\nu \leftarrow 2$
4 else
5     $\mu \leftarrow \mu \times \nu$
6     $\nu \leftarrow 2 \times \nu$
7 end if
```

The ν factor is initialized to $\nu = 2$. Notice that a series of consecutive failures results in rapidly increasing μ values.

We also need to define a *stopping criterion* for when the current solution is satisfactory. For this purpose, one could check weather the gradient is essentially **0**, that is $\mathbf{F}'(\mathbf{x}^\star) = \mathbf{g}(\mathbf{x}^\star) = \mathbf{0}$:

$$||\mathbf{g}||_\infty \leq \epsilon_1 \tag{3.1.17}$$

where ϵ_1 is a user-defined, small quantity.

Another relevant stopping criterion would be to check the gradual change in the solution of x.

$$\|\mathbf{x}_{new} - \mathbf{x}\| \leq \epsilon_2(\|\mathbf{x}\| + \epsilon_2) \tag{3.1.18}$$

This expression is a function of the $\|\mathbf{x}\|$ and thus ranges from ϵ_2^2 for small values of $\|\mathbf{x}\|$ to respectively large thresholds for large values of $\|\mathbf{x}\|$. Finally, as in all iterative processes, we need to define a safeguard to break out of infinite loop situations:

$$k \geq k_{max} \tag{3.1.19}$$

Both ϵ_2 and k_{max} the maximum allowed size of iterations are user-dependent.

3.2 GSoC Internship at MRPT - Library Design

As part of my internship in Google Summer of Code 2016 with Mobile Robot Programming Toolkit (MRPT) we developed a complete graphSLAM strategy that could manage offline datasets in the MRPT rawlog format.

The following, is a short summary of the work, so that the reader can get the necessary information without jumping to external links. Theoretical analysis of the utilized strategies are presented in sec. 3.1.

For a more information on this work refer to the following links:

- Project Discussion Page[6]
- Demo Video[7]
- Application Page[8]
- graphslam-engine technical report[9]

Goal of the internship was not to merely write one SLAM algorithm that would manage a specific version of rawlog files but rather build a complete framework using which one could:

- Experiment with new graph-based SLAM algorithms, either in the data-acquisition part (frontend) or in the optimization part (backend),
- Use MRPT built-in frontend, backend algorithms to execute graphSLAM in prerecorded offline datasets,
- Potentially execute graphSLAM in 3D datasets or in multi-robot setup

Having set these goals, we designed and programmed a set of classes that allows for a reconfigurable and extensible setup, offering the user both a set of ready-to-use graphSLAM algorithms as well as the ability to develop and integrate their own. Building blocks of the aforementioned strategy are outlined below [10].

[6] `https://github.com/MRPT/GSoC2016-discussions/issues/2`

[7] `https://www.youtube.com/watch?v=PvOyvlzrcXk&feature=youtu.be`

[8] `http://www.mrpt.org/list-of-mrpt-apps/application-graphslamengine/`

[9] `https://www.dropbox.com/s/u7phs612qf1l8bb/graphslam-engine-guide.pdf?dl=0`

[10] mrpt-graphslam library design splits the *frontend* into a node registration scheme and an edge registration scheme. The former algorithm adds new nodes in the graph (this includes the corresponding constraint between the last and the second to last inserted nodes) while the latter is the algorithm that adds edges between non-consecutive nodes in the graph

`CGraphSlamEngine`[11]
Class that manages the overall execution of graphSLAM.

`CRegistrationDeciderOrOptimizer`[12]
Common parent of all decider/optimizer classes.

`CNodeRegistrationDecider (NRD)`[13]
Node registration decider schemes add nodes to the graph according to a specific criterion and should implement the CNodeRegistrationDecider abstract class. The latter provides the basic methods that have to exist which are called from the main `CGraphSlamEngine` instance.

For an example of inheriting from this class see `CFixedIntervalsNRD`[14]

Currently two specific node registration schemes have been implemented:

- `CFixedIntervalsNRD`[15]
  Decider registers a new node in the graph if the distance or the angle difference with regards to the previous registered node surpasses a corresponding fixed threshold. Decider makes use only of the `CObservationOdometry` instances in the rawlog file.

- `CICPCriteriaNRD`[16]
  Decider registers a new node in the graph if the distance or the angle difference with regards to the previous registered node surpasses a corresponding fixed threshold. Decider measures the distance from the current position to the previous registered node using ICP (i.e., finds the relative pose transformation by matching the current range scan against the range scan of the previous node). In case of noisy 2D laser scans, decider can also use odometry information,if available, to locally correct and smoothen the robot trajectory. Decider makes use of `CObservation2DRangeScans` or `CObservation3DRangeScans` instances. More on the Iterative Closest Point algorithm (ICP) is given in the (sec. 3.1.1).

Note: As a naming convention, all the implemented node registration deciders are suffixed with the NRD acronym.

`CEdgeRegistrationDecider` (ERD)[17]
Edge registration decider schemes add edges between already added nodes in the graph according to a specific criterion and should implement the CEdgeRegistrationDecider abstract class. The latter provides the basic methods that have to exist which are called from the main `CGraphSlamEngine` instance.

For an example of inheriting from this class see `CICPCriteriaERD`[18].

Currently two specific edge registration schemes have been implemented:

- `CICPCriteriaERD`[19]

[14]`http://reference.mrpt.org/devel/classmrpt_1_1graphslam_1_1deciders_1_1_c_fixed_intervals_n_r_d.html`
[15]`http://reference.mrpt.org/devel/classmrpt_1_1graphslam_1_1deciders_1_1_c_fixed_intervals_n_r_d.html`
[16]`http://reference.mrpt.org/devel/structmrpt_1_1graphslam_1_1deciders_1_1_c_i_c_p_criteria_n_r_d_1_1_t_params.html`
[18]`http://reference.mrpt.org/devel/classmrpt_1_1graphslam_1_1deciders_1_1_c_i_c_p_criteria_e_r_d.html`
[19]`http://reference.mrpt.org/devel/structmrpt_1_1graphslam_1_1deciders_1_1_c_i_c_p_criteria_e_r_d_1_1_t_params.html`

Register new edges in the graph with the last inserted node. Criterion for adding new edges should be the goodness of the candidate ICP edge. The nodes for ICP are picked based on the distance from the last inserted node. Decider makes use of 2DRangeScans or 3DRangeScans.

- `CLoopCloserERD`[20]
  Evaluate sets of potential loop closure edges in the graph based on their pairwise consistency matrix.Decider first splits the graph into partitions based on the 2D laser scans of the nodes and then searches for potential loop closure edges within the partitions. Goal is to register only a subset of the potential loop closure edges that maximally agree with each other. Decider is implemented based on [58], [40]. A more detailed approach into the loop closure scheme is given in sec. 3.1.2.

Note: As a naming convention, all the implemented edge registration deciders are suffixed with the ERD acronym.

`CGraphSlamOptimizer` (GSO)[21]

Optimizer classes optimize an already constructed graph so that the registered edges maximally agree with each other. They should implement the methods defined in the `CGraphSlamOptimizer` abstract class.

For an example of inheriting from this class see `CLevMarqGSO`[22].

Note: As a naming convention, all the implemented optimizer classes are suffixed with the GSO acronym.

In fig. 3.2.1 a visual representation of the aforementioned classes structure is given.

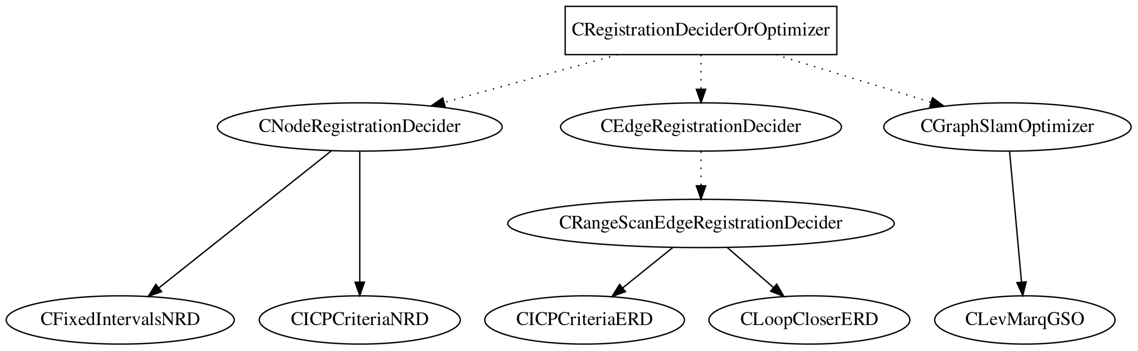


Figure 3.2.1: Hierarchy of graphslam-lib classes. Notice that all deciders/optimizers inherit from a common parent `CRegistrationDeciderOrOptimizer` which sets the overall behavior of all decider/optimizer classes (e.g., basic method calls, initialization of logging instances) Also notice the `CRangeScanEdgeRegistrationDecider` interface as a middleware between the ERD interface and the actual implementations. This comes in handy to define methods and variables available to all range scan-based deciders which may not be applicable though to deciders utilizing other measurements (e.g., camera).

[20]`http://reference.mrpt.org/devel/classmrpt_1_1graphslam_1_1deciders_1_1_c_loop_closer_e_r_d.html`

[22]`http://reference.mrpt.org/devel/classmrpt_1_1graphslam_1_1optimizers_1_1_c_lev_marq_g_s_o.html`

3.3 Application interface

Users of the MRPT graphslam-engine application can make use of the corresponding graphical user interface (GUI) to debug, interact and in general inspect the ongoing SLAM operation. As a standard MRPT application the GUI was built on top of OpenGL[23] and wxWidgets[24][25].

As of the current implementation, the `CGraphSlamEngine` main class is responsible of the visualization of the following in the GUI window:

Running time
Time is either imposed by the parsed dataset or, Wall-clock time if running online (see online wrapper in sec. 3.5)

Names of used deciders/optimizer

Robot odometry path

Optimized Trajectory
This is essentially a series of node positions as computed after the graph optimization.

Ground-truth Trajectory
If the ground-truth trajectory of the robot agent is available, it will also be displayed for reference. Users can currently support this via an external textfile.

Latest laser scan measurement

Rough estimation of generated map
An estimation of the generated map is also displayed as a cloud of points based on the cached measurements and the optimized positions from which these measurements were recorded at.

Overall SLAM execution statistics
Among others, these statistics include

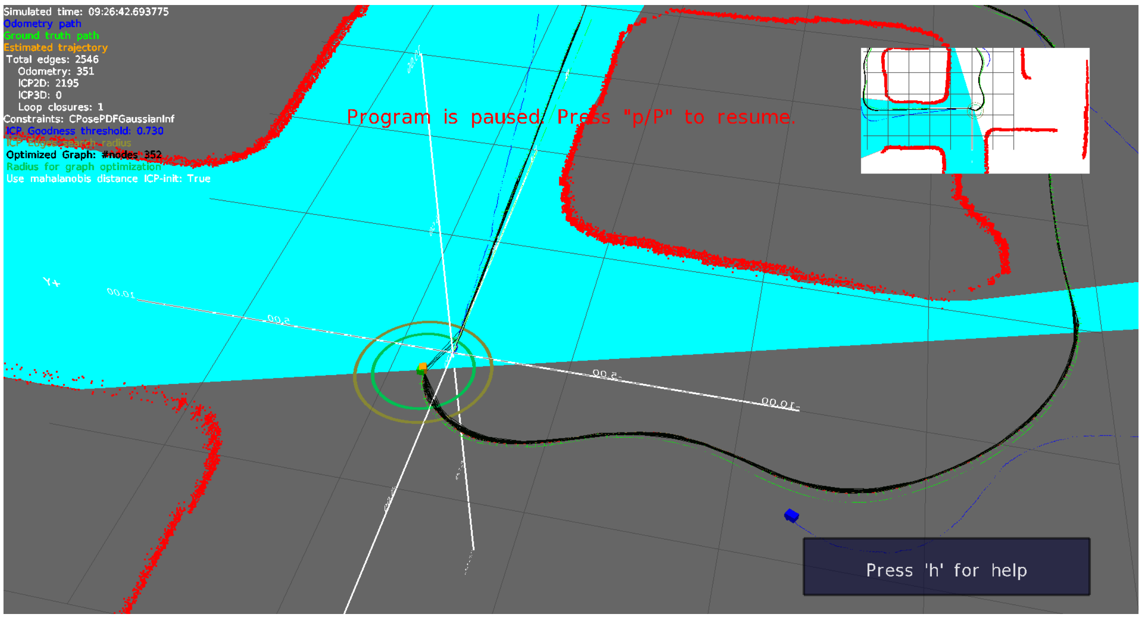
- Total number of nodes
- Total number of edges as well as number of edges based on the type of sensor they were registered by (e.g., odometry, 2D scan, 3D scan).
- Loop closure edges

Additionally, the graphslam-engine application offers hotkeys to toggle any of the visualized objects in the window on the fly (without restarting process). Finally, the `CRegistrationDeciderOrOptimizer` class interface provides methods to

- Add new visualization features
- Associate hotkeys at which the implemented decider/optimizer will listen to and be notified when one is pressed

This way user-defined decider/optimizer classes can also benefit and add visuals of their own. Examples of implemented visual features along with their corresponding descriptions are presented in fig. 3.3.1, fig. 3.3.2

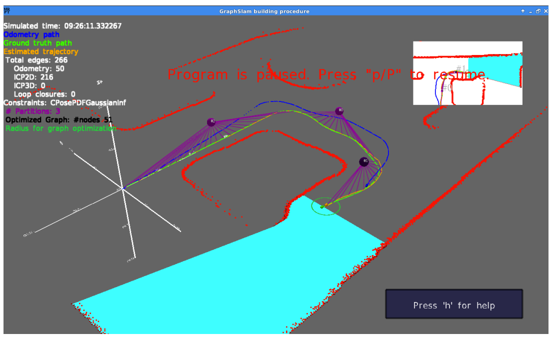
[23] `https://www.opengl.org`
[24] `https://www.wxwidgets.org`
[25] As of the time of writing, MRPT developers are looking to transition to the Qt GUI Library[26]



```
Simulated time: 09:26:42.693775
Odometry path
Ground truth path
Estimated trajectory
 Total edges: 2546
   Odometry: 351
   ICP2D: 2195
   ICP3D: 0
   Loop closures: 1
Constraints: CPosePDFGaussianInf
 ICP Goodness threshold: 0.730
 ICP Edges search radius
 Optimized Graph: #nodes 352
 Radius for graph optimization
 Use mahalanobis distance ICP-init: True
```

```
Program is paused. Press "p/P" to resume.
```

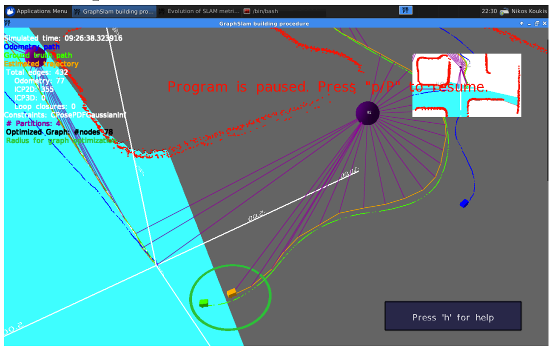
(a) View of features initially added by the `CGraphSlamEngine`. On the left, running time, SLAM-related statistics and current constraint type is provided. In the main window, the current laser scan is depicted in cyan while the blue, orange and green lines correspond to the odometry, SLAM and ground-truth trajectory respectively. Additionally the underlying graph is shown in a series of black lines while the generated map as a cloud of (red) points.



```
Simulated time: 09:26:11.332267
Odometry path
Ground truth path
Estimated trajectory
 Total edges: 266
   Odometry: 50
   ICP2D: 216
   ICP3D: 0
   Loop closures: 0
Constraints: CPosePDFGaussianInf
 # Partitions: 3
 Optimized Graph: #nodes 51
 Radius for graph optimization
```

```
Program is paused. Press "p/P" to resume.
```

(b) Graph partitioning. This is an essential step in the detection of potential loop closures since the latter may arise in groups that contain nodes with large nodeID difference. Visualization of groups is critical in the inspection of this procedure



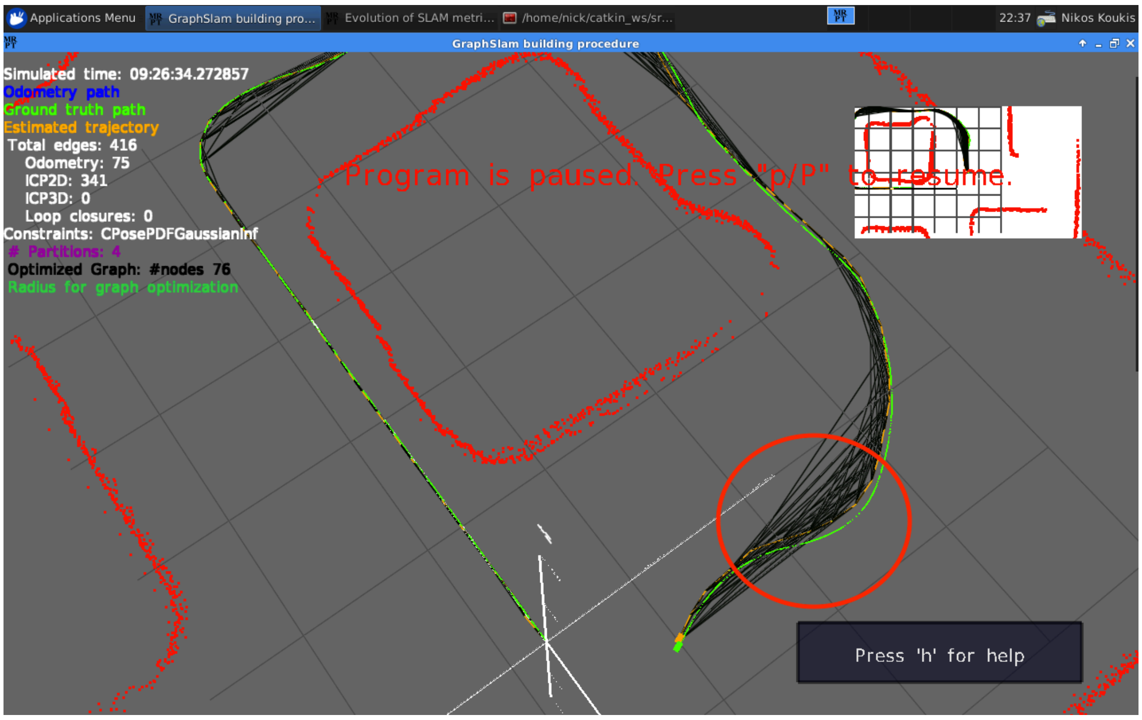
```
Simulated time: 09:26:38.323916
Odometry path
Ground truth path
Estimated trajectory
 Total edges: 432
   Odometry: 77
   ICP2D: 355
   ICP3D: 0
   Loop closures: 0
Constraints: CPosePDFGaussianInf
 # Partitions: 4
 Optimized Graph: #nodes 78
 Radius for graph optimization
```

```
Program is paused. Press "p/P" to resume.
```

Applications Menu
GraphSlam building pro...
Evolution of SLAM metr...
/bin/bash
22:30
Nikos Koukis
GraphSlam building procedure
Press 'h' for help

(c) Partitioning scheme has detected potential loop closures Initial and last inserted nodes are placed in the same partition

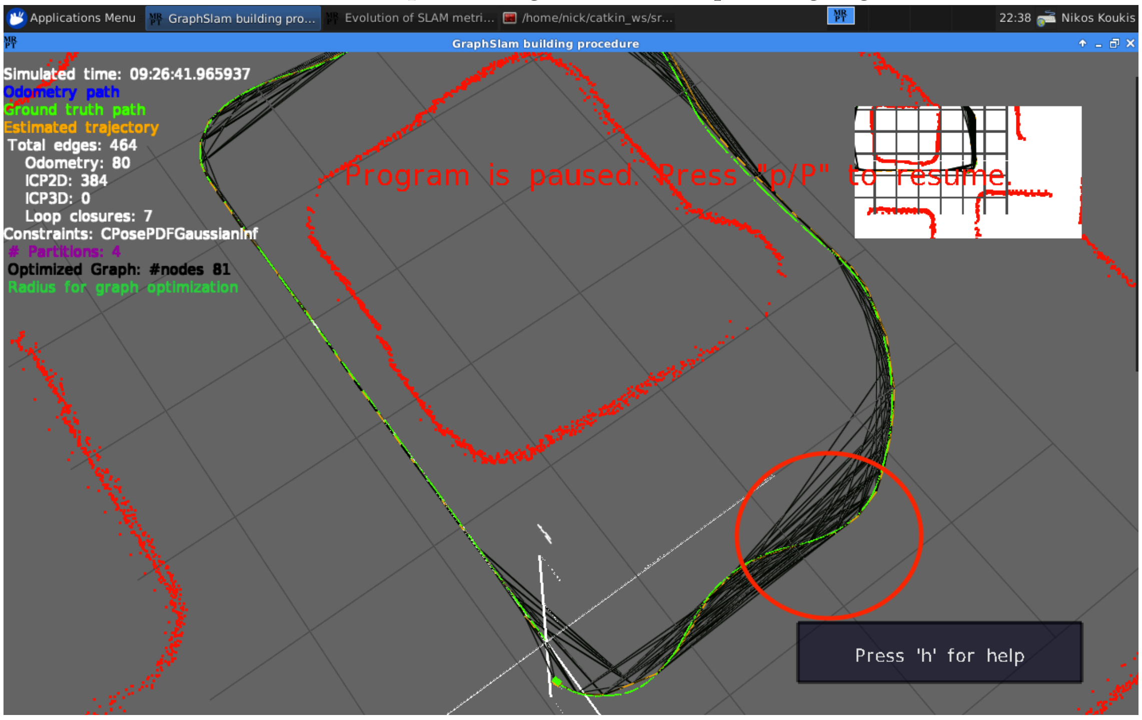
Figure 3.3.1: Examples of visualization features picked from various parts of the mrpt-graphslam lib library.


Applications Menu
GraphSlam building pro...
Evolution of SLAM metri...
/home/nick/catkin_ws/sr...
22:37
Nikos Koukis
GraphSlam building procedure
Press 'h' for help

```
Simulated time: 09:26:34.272857
Odometry path
Ground truth path
Estimated trajectory
 Total edges: 416
    Odometry: 75
    ICP2D: 341
    ICP3D: 0
    Loop closures: 0
Constraints: CPosePDFGaussianInf
 # Partitions: 4
 Optimized Graph: #nodes 76
 Radius for graph optimization
```

```
Program is paused. Press "p/P" to resume.
```

(a) Situation **prior** to registration of loop-closing edge


Applications Menu
GraphSlam building pro...
Evolution of SLAM metri...
/home/nick/catkin_ws/sr...
22:38
Nikos Koukis
GraphSlam building procedure
Press 'h' for help

```
Simulated time: 09:26:41.965937
Odometry path
Ground truth path
Estimated trajectory
 Total edges: 464
    Odometry: 80
    ICP2D: 384
    ICP3D: 0
    Loop closures: 7
Constraints: CPosePDFGaussianInf
 # Partitions: 4
 Optimized Graph: #nodes 81
 Radius for graph optimization
```

```
Program is paused. Press "p/P" to resume.
```

(b) Situation after registration of loop-closing edge. Notice how the black lines of the graph, along with the estimated trajectory coincide with the ground-truth path

Figure 3.3.2: Effect of registering a loop closing edge in the graph

3.4 SLAM evaluation metric

3.4.1 Overview of metric - SLAM benchmarking

to measure of the performance of the SLAM process, during both simulations as well as possibly real-time experiments, a benchmark needs to be defined. For this purpose we utilize a SLAM evaluation metric initially proposed in [62] [63]. Proposed metric instead of evaluating the output of SLAM algorithm as a whole (map and trajectory) utilizes only the SLAM estimated trajectory and compares it to the ground-truth trajectory or an estimation of the latter. This approach offers multiple advantages including:

Invariant to map representation
Since we don't utilize map information explicitly, the SLAM algorithm is free to choose its map representation, be it feature-based, an occupancy grid-map, or a view-based one.

Invariant to the SLAM technique
Since the metric under discussion only utilises trajectory information, it is invariant to the actual algorithm we use to solve the problem (e.g., EKF, PF, Graph-based). That is due to the fact that all algorithms provide an estimate of the trajectory (which is not the case for the environment/map estimation though). Based on this fact, it can be used, as it is suggested, to benchmark the accuracy of the different kinds of SLAM techniques as well.

3.4.2 Formulation of metric

Let $x_{1:T}$ be the poses of the estimated robot trajectory as given by the SLAM algorithm for the timesteps 1 up to T, for which $x_t \in SE(2)$ or $x_t \in SE(3)$ depending on whether its a 2D or 3D SLAM operation respectively. Additionally we define $x^\star_{1:T}$ as the reference poses of the robot, that is the ground-truth path. A straightforward way of comparing the SLAM trajectory to the ground-truth, is to utilize the following error metric:

$$\epsilon(x_{1:T}) = \sum_{t=1}^{T} (x_t \ominus x^\star_t)^2 \tag{3.4.1}$$

In eqn. 3.4.1 $\oplus, \ominus$ are the standard transformation and inverse transformation compositions respectively. We can also define $\delta_{i,j}$ as the relative transformation that connects poses of x_i and x_j $(x_i \rightarrow x_j)$[27]. Using the above, we can formulate eqn. 3.4.1 as follows:

$$\epsilon(x_{1:T}) = \sum_{t=1}^{T} \left((x_1 \oplus \delta_{1,2} \oplus \cdots \oplus \delta_{t-1,t}) \ominus (x^\star_1 \oplus \delta^\star_{1,2} \oplus \cdots \oplus \delta^\star_{t-1,t}) \right)^2 \tag{3.4.2}$$

However, as authors of [63] have claimed, this metric is suboptimal for measuring the performance of a SLAM problem. This is because an error in the estimated SLAM trajectory is propagated to all following poses considered in this analysis even when the robot motion is locally accurate.[28] An example of using this metric is illustrated in eqn. 3.4.1.

[27]Same goes for the starred version, $\delta^\star_{i,j}$

[28]By local accuracy we mean the relative transformation between two consecutive poses of the SLAM trajectory compared to the corresponding transform between the corresponding poses of the ground-truth trajectory.

Assume that the robot has estimated accurately its trajectory, except of a rotational error somewhere along the line, for example in the middle. Both prior and after the error, the robot computes the transformation between consecutive poses accurately. However, when employing the metric of eqn. 3.4.1 the error increases as more nodes after the error are considered. Thus the error depends on *the point in time* that the robot made estimation error without considering that it might not introduce any (further) error. The reason for this is that the metric at hand, computes the error in global coordinates and considers the trajectories as rigid bodies that have to be maximally aligned. In contrast, the authors propose a metric based on the *deformation energy* that is needed to transfer the estimated trajectory into the ground truth and by considering the relative displacement between poses. That is, instead of comparing x to $x^\star$ in the global reference frame, we operate based on δ and $\delta^\star$ as follows:

$$\begin{aligned}\epsilon(\delta) &= \frac{1}{N}\sum_{i,j}(\delta_{i,j} \ominus \delta^\star_{i,j})^2 \\ &= \frac{1}{N}\sum_{i,j}\left(trans(\delta_{i,j} \ominus \delta^\star_{i,j})^2 + rot(\delta_{i,j} \ominus \delta^\star_{i,j})\right)^2, \qquad (3.4.3)\end{aligned}$$

where,

N: number of relative relations

$trans(\cdot)$: Translation component of transform

$rot(\cdot)$: Rotational component of transform

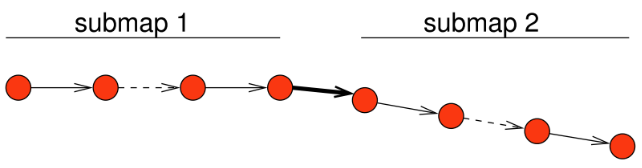
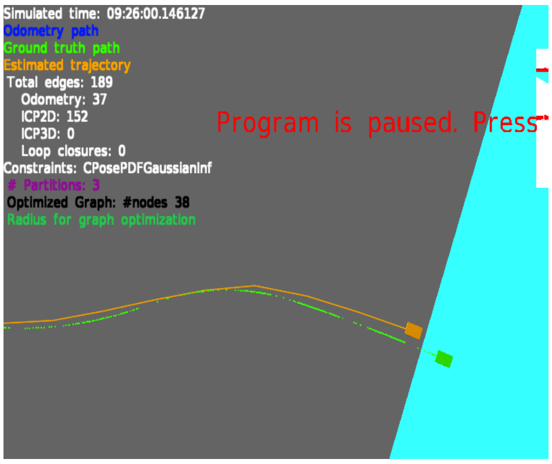


Figure 3.4.1: The robot moves along a straight line and after n poses, it makes a small angular error (bold arrow) but then continues without any further error. Both parts (labeled submap 1 and submap 2) are perfectly mapped and only the connection between both submaps contains an error. According to eqn. 3.4.1, the error of this estimates *increases with every node* added to submap 2 although the submap itself is perfectly estimated. Thus, the error depends on the point in time where the robot made an estimation error without considering that it might not introduce any (further) error. However, by using the metric of eqn. 3.4.3 the error is accounted for only once.

The algorithm, as was defined in eqn. 3.4.3 doesn't specify which poses to consider when evaluating the relative displacements $\delta_{i,j}$. As an example one could use nearby (even consecutive) poses to compute the δs

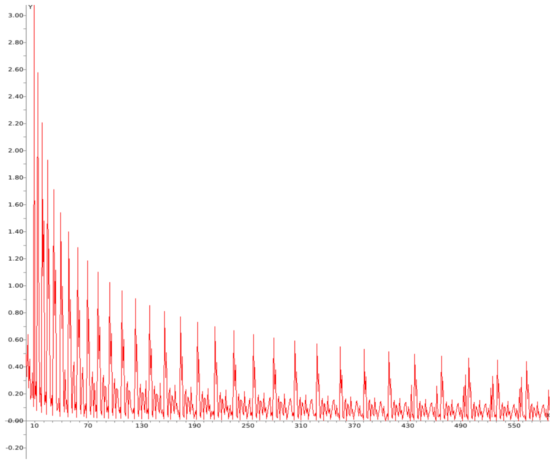
It should be noted that the metric presented here also has drawbacks. The most notable is that it only evaluates the mean estimate of the SLAM algorithm and does not consider its estimate of the uncertainty. Thus, it fails to align with the probabilistic context the whole SLAM algorithm is set in.

An example of using it during an execution of the graphslam-engine application is also illustrated in fig. 3.4.2 and fig. 3.4.3. We should remark that this is an inherent, to the application, feature that was executed *simultaneously* to the SLAM algorithm, and not an offline calculation. Users have the option of toggling the visualization (as well as the actual metric computation) ON or OFF.



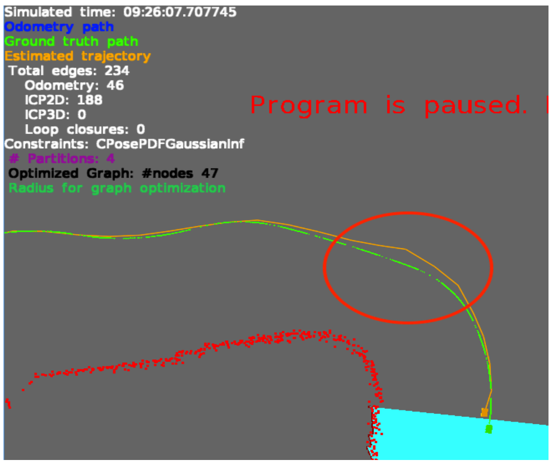
```
Simulated time: 09:26:00.146127
Odometry path
Ground truth path
Estimated trajectory
Total edges: 189
   Odometry: 37
   ICP2D: 152
   ICP3D: 0
   Loop closures: 0
Constraints: CPosePDFGaussianInf
# Partitions: 3
Optimized Graph: #nodes 38
Radius for graph optimization
Program is paused. Press
```

(a) Estimated and Ground-Truth trajectory.



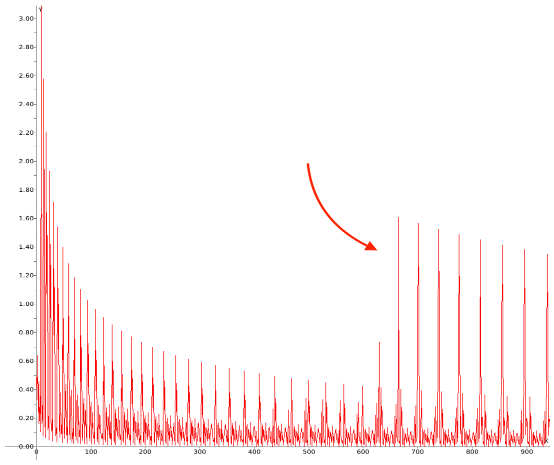
(b) SLAM metric evaluation.

Figure 3.4.2: Situation prior to divergence.



```
Simulated time: 09:26:07.707745
Odometry path
Ground truth path
Estimated trajectory
Total edges: 234
   Odometry: 46
   ICP2D: 188
   ICP3D: 0
   Loop closures: 0
Constraints: CPosePDFGaussianInf
# Partitions: 4
Optimized Graph: #nodes 47
Radius for graph optimization
Program is paused.
```

(a) Estimated and Ground-Truth trajectory.



(b) SLAM metric evaluation.

Figure 3.4.3: Situation after divergence. Notice the zig-zag recorded at the end of the estimated robot trajectory. This has a severe effect (large spikes towards the right) on the SLAM metric which however adds this error *only once*, and doesn't reconsider this for the rest of the pairs of ground-truth to estimated trajectory poses. We come to this conclusion since the error doesn't continue upwards but reduces with the same rate as prior the wrong registration.

3.5 Wrappers for usage in ROS - Online SLAM

Having successfully executed graphSLAM in multiple offline runs and simulated datasets, we then developed wrapper code to utilize the mrpt-graphslam library in native ROS applications, thus, make it possible to run graphSLAM in an online fashion (in parallel to data-acquisition from hardware sensors). Code for configuring an arbitrary robot agent is available in the csl_hw_setup[29] ROS package. Functionality is provided via a series of launchfiles, along with which, full documentation and instructions on using each one is given.

3.5.1 Configuring a single SLAM agent

Our goal is to setup a single agent as part of an existing team of graphSLAM agents as fast as possible and in an easy-to-grasp manner. By setting up a SLAM agent, configuration for the following, among other tasks, should be available:

- Setup the network communication for the agent to exchange information with a central introspection node as well as with the other agents in multi-robot graphSLAM setups.
- Start the various processes that are needed to control the robot movement (e.g., motor drivers, teleoperation nodes). Design should take in account that different robots use different processes for e.g., robot movement (Youbot drivers, Pioneer ARIA drivers) and some even custom ones (Poulias drivers for the modified Pioneer models).
- Start the processes that read information off hardware sensors (e.g., camera, laser scans). Potential design should take in account that different types of sensors may be used (e.g., SICK laser scanner instead of Hokuyo).
- Various utility ROS nodes (e.g., rosbag-player node for recording the measurements).

To do that we have used a mixture of shell environment-setting configuration files and ROS launchfiles which can be summarized as follows:

1. Create (or use one of the preexisting) configuration files. This file contains directives for starting up all the necessary ROS nodes in a single SLAM agent. An example of such configuration file that was used during the single-robot experiments for a youbot is shown in listing lst. 3.1
2. Source the aforementioned config file, either in the current shell or from the robot's `~/.bashrc` file.

   ```
   source /path/to/config/file.sh # Source it
   echo <variable defined in latter file> # Make sure that file is
   sourced correctly
   ```

3. Run corresponding *porcelain* launchfiles [30] according to the robot agent's task:

   ```
   roslaunch csl_hw_setup setup_graphSLAM_agent.launch
   roslaunch csl_hw_setup run_graphslam_real.launch
   ```

[29] `www.github.com/bergercookie/csl_mr_slam/csl_hw_setup`

[30] In the `csl_hw_setup` package, the launchfiles are split into porcelain and plumbing launchfiles Porcelain directory launchfiles (as in git-porcelain commands) are to be launched *directly* by the ROS user to setup/execute single or multi-robot SLAM. To do that they utilize launchfiles found in the corresponding plumbing/ directory. Plumbing directory, on the other hand (as in git-plumbing commands), contains roslaunch files whose task is to actually start up the corresponding ROS nodes. Each one of these files has a very specific, low-level, goal (e.g., start the Hokuyo laser node, given a set of ROS Server parameters).

We should remark that the same launchfile `run_graphslam_real.launch` is used for configuring the agent for both single and multi-robot graphSLAM. Behavior differs based on the selected NRD, ERD, GSO classes issued in the shell environment configuration file which is sourced prior to the launchfile call.

```bash
#!/usr/bin/env bash

# Tue Nov 8 10:25:04 EET 2016, Nikos Koukis
# Single robot graphSLAM
# Current configuration file was used during single-robot real-time
    testing of
# of the graphSLAM algorithm
#
# Source this file on both the ubuntu machine and youbot
#########################################################

export MR_USE_MULTIMASTER=0

export MR_IS_MULTIROBOT_GRAPHSLAM=0
export MR_DISABLE_MRPT_VISUALS=0

# This is used only in the case the Poulias Pioneer drivers are used.
    No need
# to define it otherwise
#
# Available options:
# - pioneer_3dx # yeah, its 2dx, but not worth changing code for this..
# - pioneer_2at
# - youbot
export MR_ROBOT_MODEL="youbot"

# ROS namespace under which all nodes are started
# NOTE: Previous convention was that the namespace would be
# ${MR_ROBOT_MODEL}_${MR_ROBOT_ID} but this was not handy in multi-
    robot
# communication proc
export MR_ROBOT_NS="$(hostname)"

# Record topics on current host?
export MR_RECORD_TOPICS=1

# Robot HW Drivers
# Available options are:
# - youbot
# - pioneer_poulias
# - aria
export MR_ROBOT_DRIVERS_NAME="youbot"

# All nodes read this variable and output the messages accordingly
export MR_OUTPUT_MESSAGES_TO="screen"

# LaserScanner
export MR_USE_LASER=1

# Available options are:
# - hokuyo
# - sick
export MR_LASER_NAME="hokuyo"
```

```
# [!] WARNING
# urg_node doesn't seem to work properly with the hokuyo laser scanners
    available in the Control Systems lab
# It is *strongly* advised to set this to 0
export MR_LASER_USE_URG_NODE_PKG=0 # Use the urg_node ROS pkg instead
    of the hokuyo_node

# Option only available with the urg_node laserScans package - ignored
    otherwise
export MR_LASER_SKIP_NUM_MESSAGES="1"

export MR_LASER_PORT="/dev/ttyACM0"

# Camera
export MR_USE_ONBOARD_CAMERA=0
export MR_ONBOARD_CAMERA_PORT="/dev/video0"

# Teleoperation
export MR_USE_JOYSTICK=1
export MR_JOYSTICK_PORT="/dev/input/js0"
# Available options are:
# - poulias -- Custom poulias teleoperation
# - non-holonomic
# - holonomic
export MR_JOYSTICK_CONFIG_FNAME="non-holonomic" # configuration files
    are found in $(rospack find csl_common)/config/joystick
# Available options are:
# - poulias
# - generic
export MR_JOYSTICK_FOR="generic"

# Marker IDs for  common origin and ground-truth paths
# These should exist even when no ground truth is acquired. This is
    because the
# names of the frames of the robots are initialized based on these.
export MR_ORIGIN_MARKER_ID="mf7"
# Just for initializing the robot path at anchor_frame_ID - make sure
    this is unique for every running agent
export MR_ROBOT_MARKER_ID="mf1"

# if true, Odd Aruco markers are used for tracking the agents' ground
    truth
# paths while even aruco markers for inter-robot meetings
export MR_USE_ODD_ARUCO_MARKERS_FOR_GT=1 # always needed.

# Read by the computer handling the ground-truth cameras - If True the
    cameras
# are initialized so that we have an estimation of the ground-truth of
    the
# robot paths
export MR_COMPUTE_GROUND_TRUTH=1

# Deciders/Optimizers to be used
export MR_NRD="CFixedIntervalsNRD"
export MR_ERD="CICPCriteriaERD"
export MR_GSO="CLevMarqGSO"
```

Listing 3.1: Example of a configuration file used to setup a SLAM agent

3.6 Experimental Results

To verify the behavior of the algorithm in online datasets a series of experiments have been executed. These experiments are listed in sectionssec. 3.6.3, sec. 3.6.4,and sec. 3.6.5. Each experiment section holds the hardware and software configuration, the experimental results as well as the corresponding conclusions. Prior to that, in sec. 3.6.1 the ground-truth acquisition strategy is analyzed and in sec. 3.6.2 general information regarding all three experiments are given.

3.6.1 Ground-Truth acquisition strategy

To evaluate the performance of the single robot algorithm, a rough estimate of the ground-truth trajectory of the agent must be available. For this reason, as in [64], a ground-truth acquisition system based on visual monitoring of *Aruco markers* has been developed. Instead of just moving markers and fixed origin though, current implementation includes a static aruco marker insteaed of the fixed origin.

Two *ps3 eye cameras* are placed on the ceiling of the room facing downwards [31](see fig. 3.6.1(a)). Cameras are used in combination with the *ar_sys* ROS package so that they can detect and track Aruco markers. Due to the fact that one of the cameras was free to rotate around its axes, we *did not use a static workspace origin* (as the transformation from that camera to the potential origin would have to be computed in every experiment. Instead we opted for a static Aruco marker placed between the two cameras. This way, the transformation of the cameras to the origin can be computed dynamically, every time the cameras ar_sys node detects the static marker.

The agent traversing the workspace carries another Aruco marker on top (see fig.3.6.3(b)) facing upwards. Based on this hardware configuration, we developed monitor_gt_paths.py[32]. This script automatically tracks and stores the path that *an arbitrary marker* (except the one that acts as the origin) has traversed. This way, and keeping in mind that this should be adapted to a multi-robot setup, we don't have to explicitly define, prior to the experiment, the marker IDs that are going to be used, but just make a correlation of agent to Aruco marker so that afterwards we can visualize each path.

Furthermore because we will have to account for the inter-robot data-exchange we make a distinction in the Aruco markers used, as follows [33]:

- Odd marker IDs are used for tracking the agent ground-truth path and face upwards
- Even marker IDs are used for the robot to be identified by other running agents and face towards the side of the agent they are mounted on. The ground-truth monitoring script ignores these markers.[34]

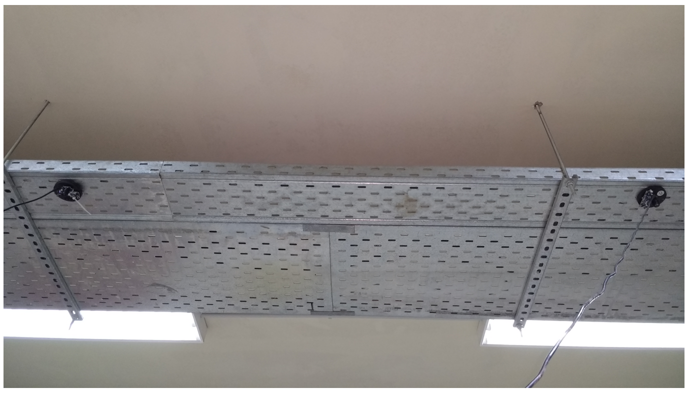
Fig. 3.6.1(a) shows a preview of the cameras configuration while fig. 3.6.2 show a preview of an Aruco marker as well as the marker that acts as the workspace origin.

[31]Placing two cameras instead of one adds to the area that the robot can traverse and thus, the area that ground-truth path can be computed.

[32]`https://github.com/bergercookie/catkin_ws/blob/master/ground_truth_fetcher/nodes/monitor_gt_paths.py`

[33]This distinction can be inversed - see the corresponding ROS parameter in the monitor_gt_paths.py script

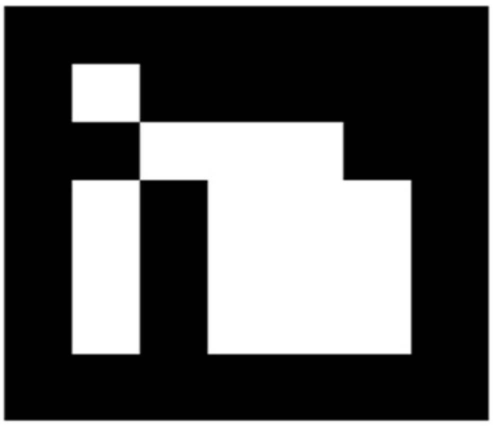
[34]Feature at hand is to be used for inter-robot meetings, a feature not yet implemented but accounted for in the overall designed process for the multi-robot graphSLAM algorithm.



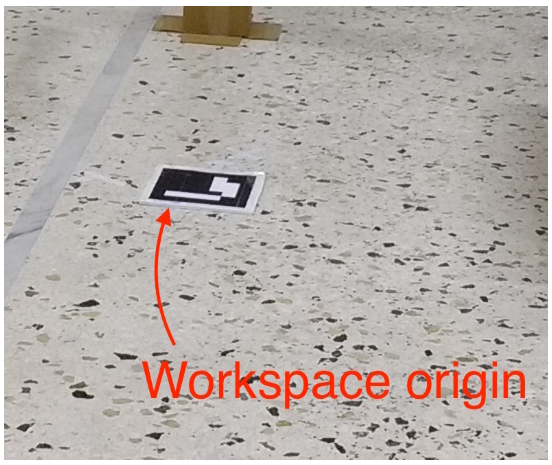
(a) Ceiling cameras used for tracking the Aruco markers. Camera on the left is free to rotate around its axis so a fixed common origin would not be appropriate.

(b) PlayStation Eye Camera

Figure 3.6.1: Ground-Truth acquisition setup



(a) Image of a typical aruco marker

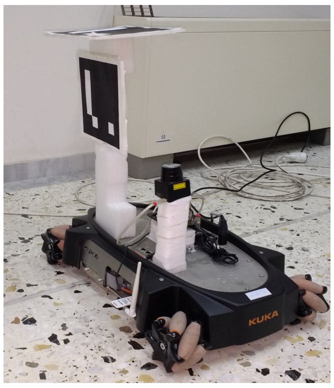


(b) Aruco marker that is used as the workspace origin

Figure 3.6.2: Aruco markers

3.6.2 Experiments - general information

In the single-robot graphSLAM experiments a *KUKA youbot* (fig. 3.6.3(a)), and a *Pioneer 2-AT*(fig. 3.6.3(b)) were used (separately). The laser scanner device used was a *Hokuyo URG-04LX-UG01* (fig. 3.6.4(a)). We opted for the Hokuyo instead of the SICK LMS 500 due to its portability and ease in usage. In the cases in which the Youbot was used due to the poor system specifications of the onboard computer, the sensor data was recorded and played back on another machine along with the corresponding ground-truth data. An overall view of the workspace that the single-robot experiments took place is given in fig. 3.6.5.


(a) The KUKA Youbot, used during the single-robot graphSLAM experiment

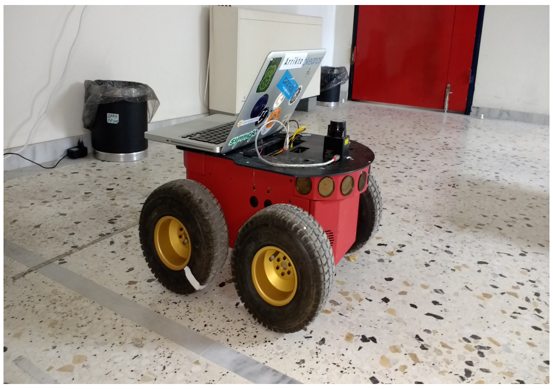

(b) The Pioneer 2-AT, used during the single-robot graphSLAM experiment

Figure 3.6.3: Robots used during overall experimentation

(a) The Hokuyo URG-04LX-UG01

(b) The SICK LMS 200

Figure 3.6.4: Laser scanner devices used during overall experimentation

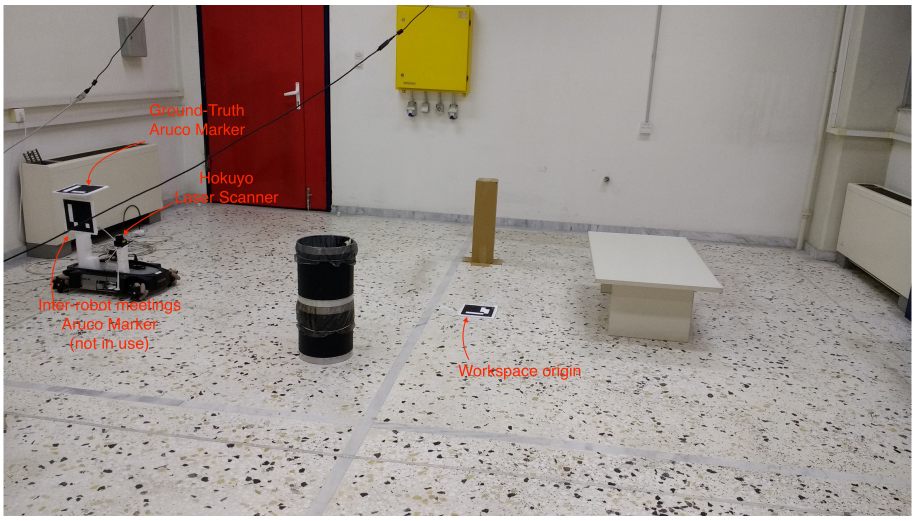

Ground-Truth Aruco Marker
Hokuyo Laser Scanner
Inter-robot meetings Aruco Marker (not in use)
Workspace origin

Figure 3.6.5: View of the workspace that the single-robot graphslam experiments were executed at. The bins, the table and the rectangular box were used as obstacles, while the Aruco markers were used for a rough estimation of the ground-truth trajectory of the robot.

Finally, videos of the experimental results are accessible here[35], while the bulk data gathered from the robot sensors and the ground-truth processes can be found from there[36]. The latter can be useful so that one can reproduce the experiment using a different software setup (e.g., use a different node registration strategy altogether). Instruction on this can be found in the mrpt_graphslam_2d application page

3.6.3 Experiment #1

Hardware configuration

Robot model: Youbot

Measurement types used: Robot odometry, laser scans

Teleoperation method: Joystick

Ground-Truth acquisition method: Monitoring of Aruco marker - see sec. 3.6.1

Miscellaneous:

- **Laser scanner device:** Hokuyo URG-04LX-UG01

Software configuration

Node Registration Decider: `CFixedIntervalsNRD`

- **Registration distance:** $0.1m$
- **Registration angle:** 5°

Edge Registration Decider: `CLoopCloserERD`

- **Use Scan Mathing** True
- **Previous Nodes for ICP:** 10
- **Eigenvalue Threshold:** 2
- **Minimum Remote Nodes to consider Loop Closure:** 2

graphSLAM Optimizer: `CLevMarqGSO`

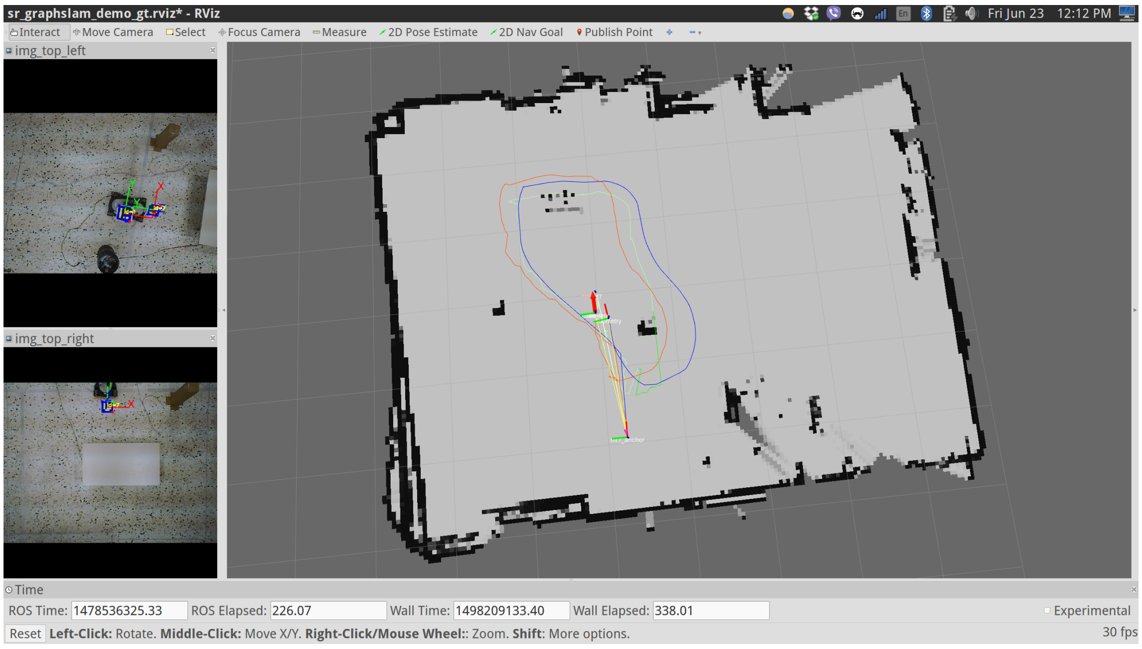
- **Optimization distance:** $0.4m$

Experimental Results

The following is the final outcome of the graphSLAM algorithm, where:

- Blue line $\rightarrow$ Odometry-only trajectory
- Green line $\rightarrow$ Ground-truth trajectory
- Orange line $\rightarrow$ graphSLAM estimated trajectory

[35] `https://www.dropbox.com/s/u7jvndtv6l7cuvo/presentation.pptx?dl=0`
[36] `https://www.dropbox.com/sh/habf5mnc5404i66/AACvbzmAQ6UrYB4ix-6aVIc9a?dl=0`



ROS Time: 1478536325.33 ROS Elapsed: 226.07 Wall Time: 1498209133.40 Wall Elapsed: 338.01

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

Figure 3.6.6: Experiment 1: graphSLAM results

Conclusions

Overall we can see that the mapping procedure is successful - there's a minor duplicate wall on the left side ofthe map, while the estimated position of the placed obstacles, by visual introspection, seems accurate. With regards to the estimated robot trajectory we can see that, especially after the turn around the table, it matches the ground-truth trajectory more closely compared to the odometry trajectory, even though the odometry divergence isn't significant due to the small overall distance [37].

3.6.4 Experiment #2

Hardware configuration

Same as in sec. 3.6.3.

Software configuration

Node Registration Decider: `CICPCriteriaNRD`

- **Registration distance:** $0.05m$
- **Registration angle:** $15°$

Edge Registration Decider: `CICPCriteriaERD`

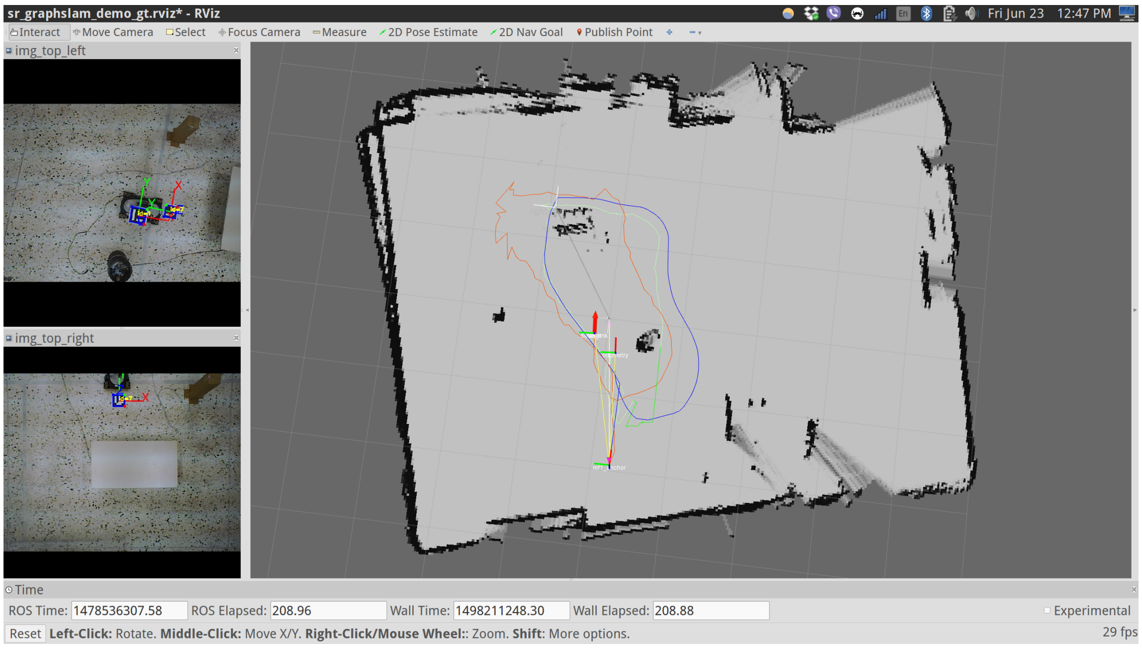
- **Distance to search for graph edges:** $2.0m$

graphSLAM Optimizer: `CLevMarqGSO`

[37]Steep straight line at the ground-truth trajectory is produced due to the robot turns around the trash can and gets out of the camera scope. Furthermore the ground-truth zig-zag accounts for minor errors in the ar_sys position estimation scheme.

- **Optimization distance:** $1.0m$

Experimental Results



ROS Time: 1478536307.58 ROS Elapsed: 208.96 Wall Time: 1498211248.30 Wall Elapsed: 208.88

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

Figure 3.6.7: Experiment 2: graphSLAM results

Conclusions

Compared to the results of sec. 3.6.3, the fine map quality remains while the estimated trajectory is still close to the ground-truth even though no odometry information was explicitly utilized. We can however detect rougher edges especially at the robot turns

3.6.5 Experiment #3

For the last single-robot experiment, a Pioneer 2-AT was used (see fig. 3.6.3(b)). This is useful for testing the algorithm mapping in larger areas, since the robot at hand has an embedded battery and thus, is capable of traversing larger distances. Due to faulty odometry data from the (custom) pioneer robot drivers[38] only the laser data was used for graphSLAM. The following sections describe the overall hardware and software configuration:

Hardware configuration

Robot model: Pioneer 2-AT

Measurement types used: Laser scans

Teleoperation method: Joystick

[38]Current Pioneer model doesn't use the Aria library and its native control system since a custom one was designed as part of Apostolos Poulias' diploma thesis in the Control Systems Lab. For more on this, see the corresponding github repository: https://github.com/bergercookie/csl_pioneer_drivers

Ground-Truth acquisition method: None

Miscellaneous

- **Laser scanner device:** Hokuyo URG-04LX-UG01

Software configuration

Node Registration Decider: `CICPCriteriaNRD`

- **Registration distance:** 0.5
- **Registration angle:** $5°$

Edge Registration Decider: `CLoopCloserERD`

- **Previous nodes to search try registering constraints with:** 10

graphSLAM Optimizer: `CLevMarqGSO`

- **Optimization distance:** $4m$

Experimental Results

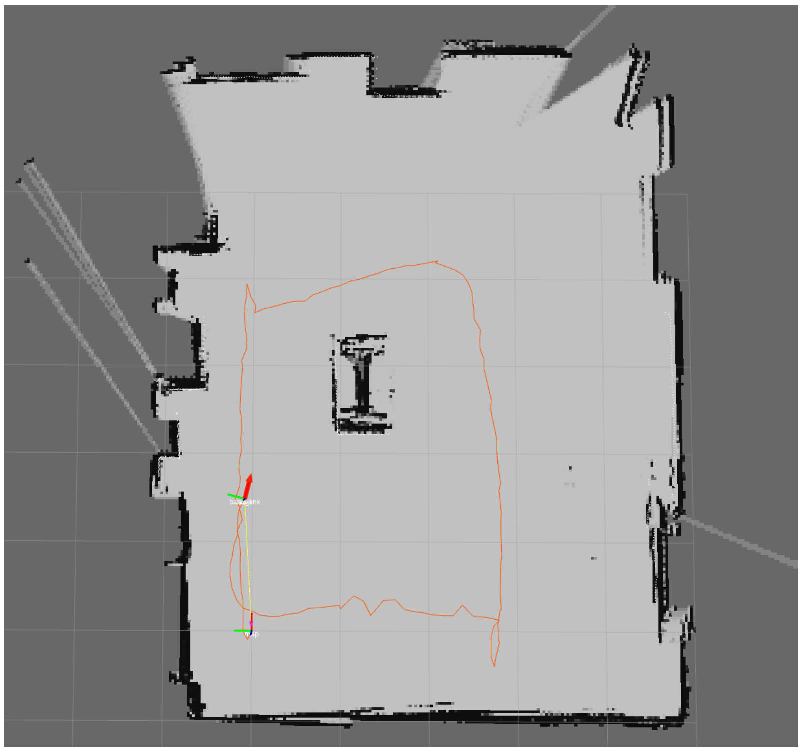


Figure 3.6.8: Experiment 3: graphSLAM results

Conclusions

Although there are minor duplicate walls (bottom), overall map is quite accurate.

Chapter 4

Inter-robot Communication

Communication between the SLAM agents is a key issue in multi-robot setups. Related strategies are mainly categorised based on the *type of information that the agents share*:

- Raw sensor data [65]

  Sensor data typically include odometry information, laser scans or camera data. This brings more flexibility for every agent to evaluate the measurements in its own way but also requires higher bandwidth, reliable communication means for the robots to use as well as redundancy of computations (each agent separately evaluates the received data).

- Processed ready-to-use data [44].

  Sharing processed data (e.g., graphs or grid maps) on the other hand, reduces the bandwidth usage and overall computational cost but the mapping procedure of each receiver robot is dependant on the accuracy of the exchanged information [3].

In our implementation we resort to a combination of the aforementioned strategies largely based on [6]. A brief overview explaining the communication procedure between two robot agents is given below.

- Each robot executes, independently, single robot graphSLAM
- When in close range they start exchanging their local maps; each robot transmits the following information upon every new node addition in its own graph:
  - Laser Scan measurement corresponding to the latest registered node in the graph
  - List of last X node positions of own graph. Due to graph optimization these positions are subject to change as new observations are being processed. During experimentation the last 10 node positions were transmitted.
- Receiver robots cache the transmitter's node positions as well as laser scans. However, as each robot doesn't know its starting position (and correspondingly its current) with regards to its neighbors, the received information cannot yet be used as they are taken with regards to the neighbor's frame of reference.

  When enough nodes are both registered in own graph and have also been received from the other agent, robot recreates other agent's graph and *tries to find a valid transformation (translation and rotation)* with which own grid map and the recreated one are best aligned. For this, a modified RANSAC scheme explained in sec. 5.1 is utilized.

- Even though the receiver can now directly add the new nodes, to simplify the integration step, it also asks the transmitter of a *condensed graph* that only contains the nodes that the receiver is going to add in its own graph. This way the receiver just has to merge the condensed local graph with its own using the grid map aligning transformation as the inter-graph constraint. After all, exchanging graph information (nodes and edges) add little overhead to the overall network usage (compared to e.g., the laser scans).

  This implementation also adds to the overall flexibility as the receiver robot can alter the received node positions after its own graph optimization step, in case they contradict with its own registered nodes.

- After first time integration, receiver has an estimation of the transformation between its own and the transmitter's graph so when it receives new data from its counterpart, it can directly incorporate them in its own graph. However, to account for optimization of the node positions on the part of the transmitter, receiver incorporates new nodes in batches (of typically 5 nodes).

This implementation offers a couple of advantages, most notably:

- Minimization of overall used bandwidth, as the only large bulk of information transmitted is the laser scan, which happens on every new node registration. Validation of this statement is provided in [6], since the information transmitted is essentially the same as in this case.

- Communication procedure is robust to limited bandwidth and is does not assume any prior network infrastructure as agents communicate having predetermined a prior ad-hoc network on which they publish their data to any available subscribers.

Chapter 5

Multi-robot SLAM algorithm

In this section we focus on extending the single-robot graphSLAM algorithm pieces (see sec. 3) so that they can be employed in a multi-robot framework. Sec. 5.1 describes a map-matching technique for deciding the relative transformation of two arbitrary robot agents while sec. 5.2 offers insight into the implementation of the communication procedure using ROS and the `multimaster_fkie` package and the `map_merger_node` application. Finally sec. 5.3 presents the simulation setup and results while sec. 5.4 presents a real-time experiment with 2 Pioneer-2dx models.

5.1 Multi-hypothesis map-matching

Current section offers insight in the grid map matching algorithm that we employ to align the map of each robot with the map that it recreated using the measurements fetched from a neighboring robot. By doing so, robot computes a rigit transformation between own and neighbor map which can be used for integrating the received nodes and measurements in its own graph. For more details on the algorithm, readers are encouraged to study the original paper [66], on which this section is heavily baed on anyway.

Algorithm comprises a special instance of generic image registration as it tries to compute a valid transform by finding correspondences between a set of sparse features derived from the grid maps. To deal with the uncertainty and ambiguity from matching grid maps, authors introduce a modified RANSAC algorithm which seraches for a dynamic number of internaly consistent subsets of feature pairings from which hypotheses about the candidate translations and rotations between the maps are derived. Finally authors provide test results from using a various combinations of detector and descriptor algorithms, which illustrate the robustness of the algorithm at hand.

5.1.1 Algorithm overview

A central idea in the current algorithm design is the *dual representation* of the evaluated maps where both occupancy grid maps as well as point maps are utilized. These maps complement each other and maintaining them only requires updating both simultaneously using the same sensory data. Based on this dual representation, algorithm consists of two discrete steps:

1. Grid maps are firstly matched without any a priory information

2. Having a rough alignment estimate from the first step, corresponding point maps are aligned so that the matching is refined.

The second step is implemented using an ICP-variant and is a well studied problem, while the first (grid-to-grid matching) is a far more challenging one since an initial estimate for the transform is not available. Based on these remarks, the rest of the section deals exclusively with the grid-to-grid matching.

We can classify image registration techniques into intensity-based and feature extraction-based. Authors have chosen to follow the second approach (feature extraction) since is computationally more efficient, thus a viable approach especially in real-time mapping scenarios.

Furthermore, instead of returning just the single best matching hypothesis, algorithm computes and reports matching hypotheses in the form of a Sum of Gaussians (SOG). This can come in handy in mapping approaches that are based on the Bayesian probabilistic framework and can utilize multiple-hypotheses (e.g., particle filtering).

An outline of the algorithm steps is given in fig. 5.1.1. A step-by-step analysis is also given below:

1. Firstly map images are preprocessed to soften out the irregularities that are commonly found in grid maps and are due to high-frequency noise from the sensors used.

2. Interest points are extracted using a certain detector scheme while their surrounding area is modelled using a descriptor. After experimenting with various combinations of detector and descriptor pairs the authors have concluded in using either the Harris or Kanade-Lucas-Tomasi detectors along with a descriptor consisting of a circular patch centered at the feature. These pairs offer the best combination of both performance maximization and reduction in computational cost.

3. Set of candidate correspondences (i.e., C) between features in both images are generated. These correspondences are determined by means of a measure of similarity between their descriptors.

4. Due to ambiguity a feature of one map may have multiple candidate features on the other. From all those candidates a modified RANSAC algorithm obtains a *subsets of internally consistent hypotheses* $C_i \in C$ by imposing *uniqueness*, meaning that each feature of one map must correspond up to just one in the other. Unlike the standard RANSAC algorithm, authors choose to keep not only the single best solution but a dynamic number of them.

5. As mentioned earlier, the potential transformation **q** between the two maps is modelled as a Sum of Gaussians (SOG) and can be written down as:

$$p(\mathbf{q}) = \sum_i N\Big(\mathbf{q}; \mathbf{q_i^\star}, \mathbf{Q_i}\Big)\omega_i \tag{5.1.1}$$

   where

   ω_i is the weight of a Gaussian kernel$\Big(\sum_i \omega_i = 1\Big)$

   $\mathbf{q_i^\star}, \mathbf{Q_i}$, is the mean and covariance matrix of that kernel

   The distribution can also be expanded using the law of total probability over all

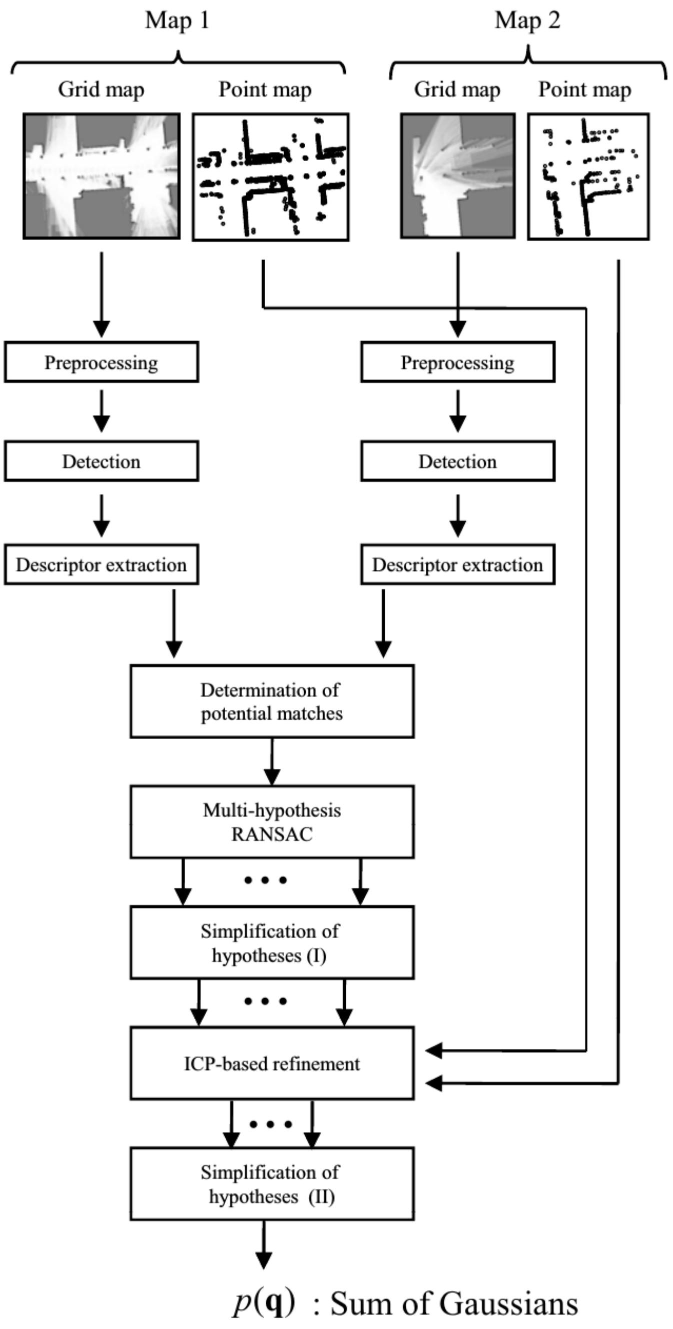


Figure 5.1.1: An overview of the proposed method for map-matching, which aligns a pair of maps each comprising a grid map and a corresponding point map. It first registers the grid maps to obtain a set of potential transformations $\mathbf{q}$ which are then refined by running ICP-based alignment on the corresponding point maps.

potential correspondences C_i as follows:

$$p(\mathbf{q}) = \sum_{\forall C_i} p\big(\mathbf{q}|C_i\big)P(C_i) \tag{5.1.2}$$

Comparing eqn. 5.1.1 and eqn. 5.1.2 it is clear that the authors choose $P(C_i)$ as the SOG weights and also, model the density of **q** given a set of correspondences as a Gaussian distribution, that is [1]:

$$p(\mathbf{q}|C_i) = N\Big((\mathbf{q}; \mathbf{q_i^\star}, \mathbf{Q_i}\Big) \tag{5.1.3}$$

6. After the RANSAC stage, the computed SOG distributions are simplified. This is an action towards reducing the cost of the upcoming refinement step, in which ICP is applied to the corresponding point maps to improve the estimate of the mean map transformation $\mathbf{q}^\star$. Finally the resulting SOG is again tested for further potential simplifications obtaining the final, possibly multi-model probability density distribution for the map transformation.

5.1.2 Feature Extractions - Detectors

The most desirable property of any detector is its *repeatability*, that is, its ability to detect a given feature when it appears in different images.

Overall the authors tested the following detector algorithms:

- The **Harris detector** which searches for points where the structure tensor [2] has two large eigenvalues.
- The **Kanade-Lucas-Tomasi (KLT) method** that also relies on the structure tensor but detects salient points where one of the eigenvalues exceeds a given threshold.
- The detection phase of the **SIFT** algorithm. This identifies scale-space extrema in pyramids of difference-of-Gaussians. Method essentially aims at detecting blobs instead of corners.
- The detection phase of the **SURF** algorithm based on the Hessian matrix.

Due to the nature of laser range scans (which is the main sensor used in our implementation) artifacts arise in the generated grid maps. If interpreted as high-frequency noise, these can be cancelled out of the preprocessed image by first applying a Gaussian filter followed by a median filter.

[1] Even though the authors have derived the parameters of eqn. 5.1.3 and have set the mathematical proofs for expressing the uncertainty of a transformation, these parts are not included in current analysis since in the our usage of the algorithm in the multi-robot SLAM, we are mostly interested in the mean of the transformation.

[2] The structure tensor, also referred to as the second-moment matrix, is a matrix derived from the gradient of a function. It summarizes the predominant directions of the gradient in a specified neighborhood of a point, and the degree to which those directions are coherent

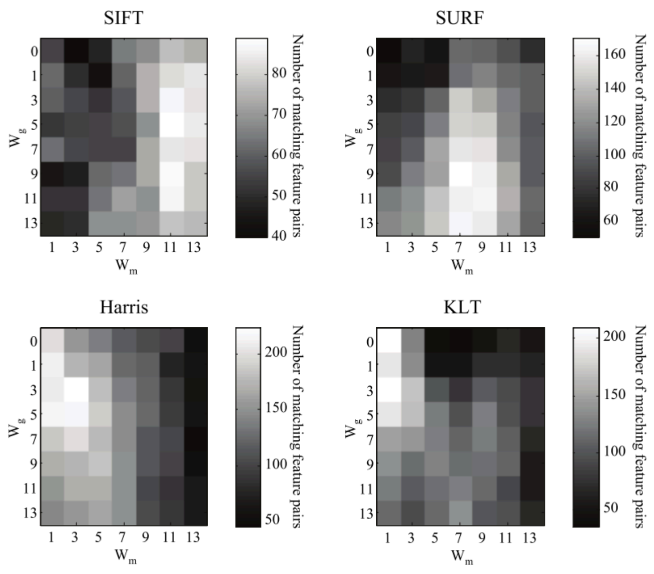
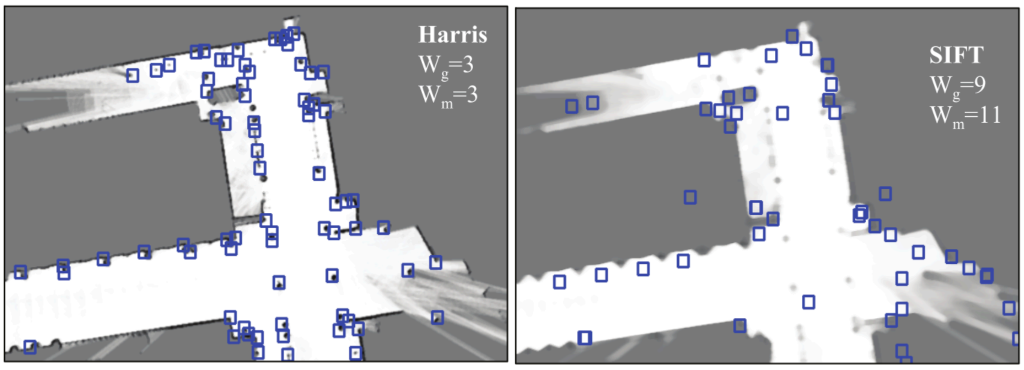


Figure 5.1.2:]
A measure of the repeatability for each detector for different sizes of the Gaussian (W_g) and the median filter (W_m). Brighter colors indicate a higher number of common features detected in both maps Notice that the Harris and KLT detectors (i.e., corner detectors) find most matchings after preprocessing with small filter sizes while the exact opposite can be said for the detector parts of the SIFT and SURF algorithms (i.e., for blob detectors)



(a) Harris detector (b) SIFT detector

Figure 5.1.3: Examples of detected features in images after the ideal, for the used detector, image preprocessing

5.1.3 Descriptors

After the detection phase, the features are assigned distinctive descriptors in order to establish correspondences. Authors have studied the usage of the following 4 descriptors:

SIFT: Method is based on histograms of image gradients, obtaining a 128-length descriptor vector

SURF: Method is based on the responses of Haar-wavelets [3].

[3]The Haar wavelet is a sequence of rescaled "square-shaped" functions which together form a wavelet

Intensity-domain spin images (Spin): A 2D histogram of intensities and distances, with the maximum a radius from the nearest point determined by the parameter R_{max}. The usage of distances (disregarding angles), makes this descriptor *invariant to rotations*.

Linear or logarithmic circular patches [4] Both (linear and logarithmic) map a circular region of radius R_{max} centered at * the interest point into a 2D matrix (the descriptor) of polar coordinates. Let this matrix be $\mathbf{f}(u, v)$ where the indices stand for different values of the distance and angle of the feature respectively. Methods depend on the extraction of a circular patch of the feature neighborhood in a representation *not invariant* to rotations (rotations just become shifts in the angle dimension (u)). The latter is also illustrated in fig. 5.1.4(b,c).

Similarity function of 2 descriptors

to compare two detected features (e.g., i, j) from two different maps, along with their corresponding descriptors, $\mathbf{f}_i^a \mathbf{f}_j^b$ we need to define a metric. Judging solely on SIFT, SURF and Spin descriptors, the most logical metric to use would be the Euclidean distance since these representations do not take the rotation into account. However this is not the case for the lin-polar and log-polar descriptors. As mentioned earlier two matching features only differ by a shift in the angular dimensions. Therefore the authors propose to measure the distance between two descriptors $\mathbf{f_i} and \mathbf{f_j}$ by their Euclidean distance given a rotation angle $\Delta\phi$:

$$d(\mathbf{f_i}, \mathbf{f_j}, \Delta\phi) = \left(\sum_u \sum_v \left| \mathbf{f_i}(u, v) - \mathbf{f_j}(u, v + \Delta\phi) \right|^2 \right)^{1/2} \tag{5.1.4}$$

where the angular polar coordinate v is taken modulo the corresponding size of the matrix.

This way we obtain a distance vector for each possible shift in orientation $\Delta\phi$. These distance vectors have pronounced minima for the true orientation when two features do really match, thus the proposed metric comes down to:

$$d(\mathbf{f_i}, \mathbf{f_j}) = min_{\Delta\phi}\, d(\mathbf{f_i}, \mathbf{f_j}, \Delta\phi) \tag{5.1.5}$$

5.1.4 Generation of correspondences- Evaluation of detector-descriptor pairs

Having set up a similarity criterion the goal is to generate the needed set of correspondences between detector/descriptor pairs for the pair of maps. Thus the ``goodness'' value, i.e., the validity, of each correspondence is to be evaluated.

The arguably simplest test for selecting matchings is using a threshold, which in our case means to accept a potential match between $\mathbf{f_i^a}, \mathbf{f_j^b}$ only if the distance $d_{i,j}$ between their descriptors is below a fixed value T_d [66]. However, this simple scheme has some drawbacks in the context of grid matching, because distance values between actually corresponding

family or basis. Wavelet analysis is similar to Fourier analysis in that it allows a target function over an interval to be represented in terms of an orthonormal basis pairs may vary in a relatively large range. Thus, any permissive threshold T_d which covers most of the good correspondences would suffer from a high rate of false positives. Authors introduce a second condition for establishing candidate pairings; the associated distance $d_{i,j}$ must be not only below the threshold T_d, but also sufficiently close to the best matching of $\mathbf{f_i^a}$ in map b, that is, the minimum of $d_{i,j}$ for all values of j. This restriction is characterized by a second threshold T_δ which states the maximum acceptable distance δ between a potential pairing and the best one, that is, $\delta_{i,j} = d_{i,j} - min_j d_{i,j}$. Notice that for the extreme case T_δ = 0 each feature will be associated to only one in the other map: the one with the closest descriptor.

Based on the above, the authors have set up a benchmark test using 10 pairs of submaps (with known ground truth) for which they compute the aforementioned thresholds T_d and T_δ. This operation executed for several combinations of the available detectors descriptors, the criterion for determining the thresholds is the minimization of the probability of misclassifying a correspondence as a valid or invalid candidate P_{err} The latter is given by:

$$\begin{aligned} Perr &= P(w)P_{err}(T_d, T_\delta | w) + P(v)P_{err}(T_d, T_\delta | v \\ &= P(w)P(d_{i,j} < T_d, d_{i,j} < T_\delta | w) + P(v)\Big[1 - P(d_{i,j} < T_d, d_{i,j} < T_\delta | v)\Big] \end{aligned} \tag{5.1.6}$$

where v, w stand for valid and wrong pairings respectively

The results of the benchmark are summarized in fig. 5.1.4 which shows the minimum classification error P_{err} attainable by each combination of feature detector and descriptor, along the associated average computation time (for one whole submap). These times include detection, descriptor extraction and distance computations, but they do not include the preprocessing filters discussed earlier. This would add an average of 10 to 200ms, with larger computational burdens associated to SIFT and SURF since they require larger filter kernels than the Harris or KLT methods.

Based on the benchmark results, using either the Harris or KLT detector along with a circular patch lin-polar descriptor seems to produce the better results in terms of reduction in computational complexity as well as increase in performance.

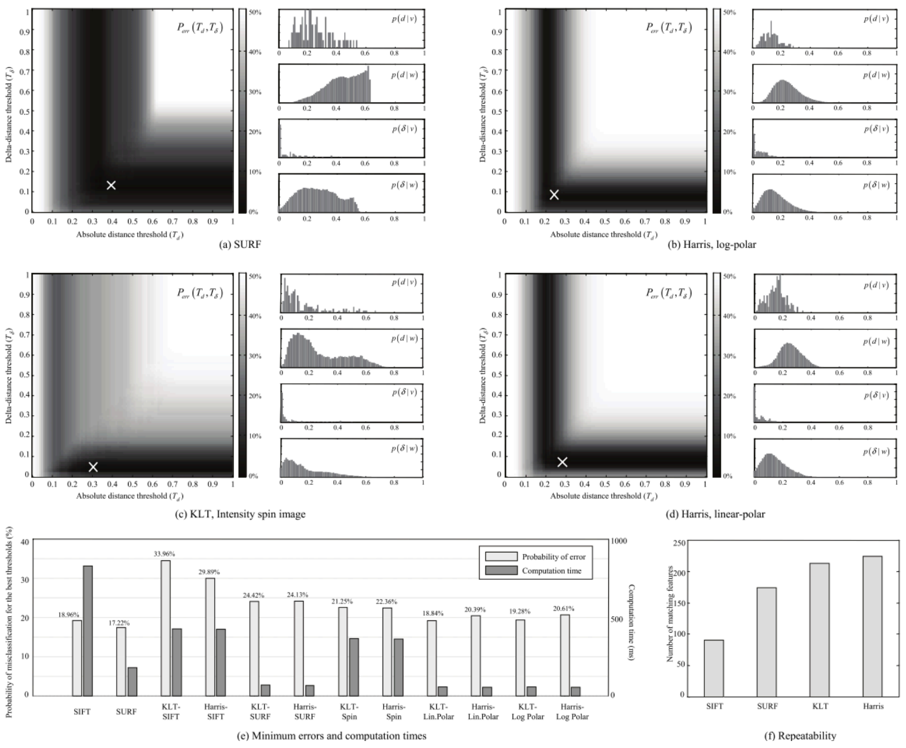


Figure 5.1.4: Benchmark results from using various detector, descriptor pairs.

5.1.5 Construction of the SOG - modified RANSAC

Final step in computing a valid transformation between the two grid maps (prior to the ICP refinement step) is to use a modified RANSAC scheme to reduce the generated pool of correspondences into subsets of self-consistent ones[5]. Due to ambiguities in the grid map matching procedure, authors choose to keep a number of valid transformations computed by RANSAC, so that the final transformation, is a Sum of Gaussians instead.

Below we provide an outline of the steps of the proposed modified RANSAC algorithm while in fig. 5.1.5 we present a pseudocode sample [66]:

- Two correspondences (minimum number to determine the distribution of the associated transformation, $p(\mathbf{q}|C_i)$) are randomly chosen from C to initialize a subset $C_i = \{c_{k1}, c_{k2}\}$.
- Uniqueness constraint is applied to the latter subset, meaning that one feature cannot appear in both correspondences c_{k1}, c_{k2} simultaneously.
- Feasibility of this pair is tested by a chi-square test which detects inconsistencies between the inter-feature spacial distance d_a and d_b measured in maps a and b respectively.
- Number of inliers supporting the proposed hypothesis $p(\mathbf{q}|C_i))$ is found for each set of initial pairings C_i. This is achieved by establishing associations between all the features in map b and those in a transformed by $\mathbf{q}$. Notice that this is a stochastic

[5]It's important to note that the map-matching module consists of a modified RANSAC scheme, i.e., a non-deterministic algorithm. Based on this, the algorithm guarantees that, given the maps under consideration have a considerable overlap, **it is likely** that it will compute the correct transformation. Furthermore in case an initial estimation of the transformation is not provided or the overlap is minimal, the probability of a wrong match increases.

```
1: $SOG \Leftarrow \varnothing$
2: $iter \Leftarrow 1$
3: repeat // $RANSAC$ iterations
4:   $\hat{\mathcal{C}} \Leftarrow \{c_{k_1}, c_{k_2}\} \subset \mathcal{C} | uniqueness(c_{k_1}, c_{k_2})$
5:   if $D^2_M(c_{k_1}, c_{k_2}) < \chi^2_{c,1}$ then // Consistency test
6:     if $\exists k | \hat{\mathcal{C}} \subset SOG_k.\hat{\mathcal{C}}$ then // Already in?
7:       // Increment the weight
8:       $SOG_k.\omega \Leftarrow SOG_k.\omega + 1$
9:     else
10:      // It is a new SOG mode
11:      $\hat{\mathcal{C}}_o \Leftarrow \hat{\mathcal{C}}$ // Save original minimal set
12:      repeat // Incorporate inliers
13:        $\hat{\mathcal{C}}_t \Leftarrow \hat{\mathcal{C}}_o \cup (i^\star, j^\star)$ // Tentative set of pairings
14:        $(q^\star_i, \mathbf{Q}^\star_i) \Leftarrow opt\_transf(\hat{\mathcal{C}}_t)$
15:        $(i^\star, j^\star) \Leftarrow \arg\max_{(i,j)} \int p_i(\xi) \tilde{p}_j(\xi) d\xi$
16:        if $D^2_M(i^\star, j^\star) < \chi^2_{c,2}$ then
17:          $\hat{\mathcal{C}} \Leftarrow \hat{\mathcal{C}} \cup (i^\star, j^\star)$ // Accept pairing
18:        end if
19:      until $D^2_M(i^\star, j^\star) \geq \chi^2_{c,2}$
20:      if $|\hat{\mathcal{C}}| \geq M$ then // Minimum inlier support
21:        // New Gaussian mode with $\omega = 1$
22:        $(q^\star_i, \mathbf{Q}^\star_i) \Leftarrow opt\_transf(\hat{\mathcal{C}})$
23:        $SOG \Leftarrow SOG \cup (\hat{\mathcal{C}}, 1, (q^\star_i, \mathbf{Q}_i))$
24:      end if
25:    end if
26:  end if
27:  $iter \Leftarrow iter + 1$
28: until $iter > maxIters$
```

Figure 5.1.5: Pseudocode of the RANSAC procedure applied to the pool of generated correspondences. For computing the uncertainty of the optional transformation, users are encouraged to read the original article [66]

data-association problem since all feature locations, and the transformation itself, have associated uncertainties. *Joint Compatibility Branch and Bound* (JCBB) is often employed to deal with the latter problem, however in this case, the exponential time complexity of JCBB make it unsuitable for use. Instead, the authors propose the sequential incorporation of matches which optimize the integral of the product of the two Gaussians, that is the *matching likelihood* of each the pairing at hand. The incorporation of inliers stops when the next best pairing candidate (i, j) has a squared Mahalanobis distance $D^2_M(i, j)$ above a given threshold $\chi^2_{c,2}$.

- The above process is repeated a number of times updated dynamically as new inliers are found. With regards to the weights of the SOG, each Gaussian mode is initially assigned a unit weight, which is incremented each time the same subset of correspondences is found in subsequent iterations (lines 6–8).
- Finally notice the existence of a minimum threshold of required inliers M for the algorithm to accept a hypothesis. This was heuristically set by the authors to 15% of the average number of features found in each map.

5.2 Implementation Insight

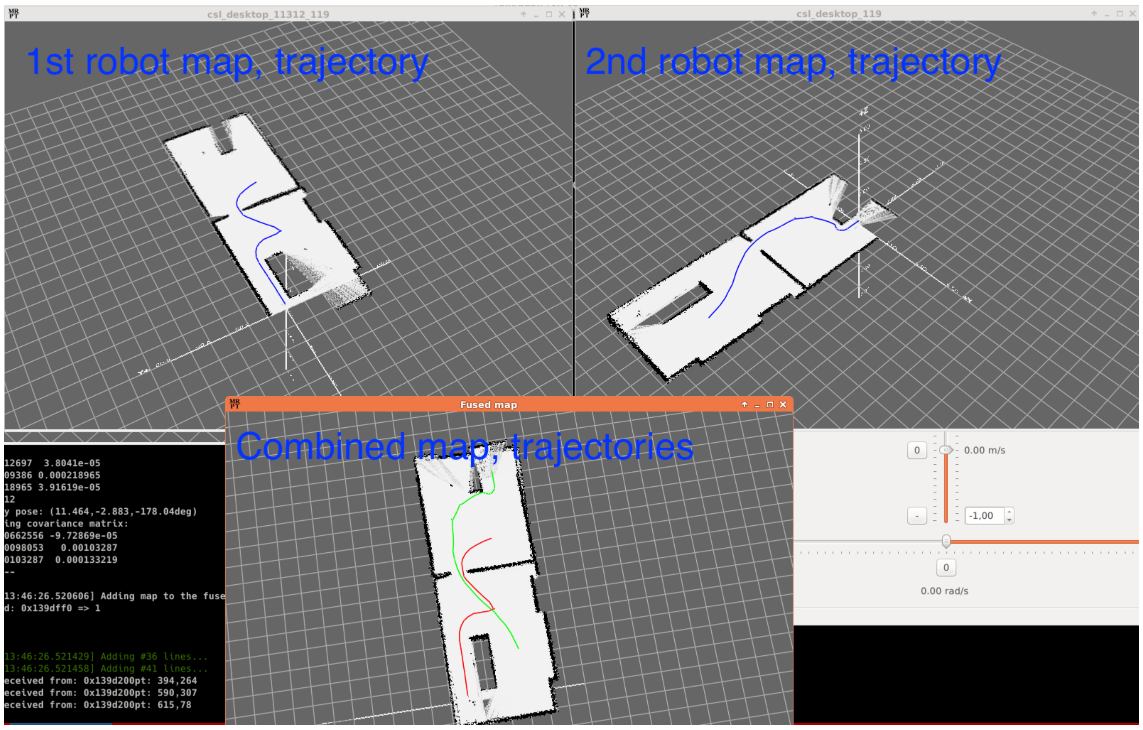
5.2.1 Communication procedure

As explained in sec. 4 communication is a key issue in multi-robot SLAM. To implement it in an efficient way and make sure that the overall algorithm is independent of the number of running SLAM agents we make use of the ROS `multimaster_fkie` package and the multicast protocol. This allows every SLAM agent to run as an independent ROS node process and on its own ROS core. This offers the following advantages:

- Using the `master_discovery` node of multimaster_fkie each ROS core dynamically finds all the other available ROS cores running in the same network that it runs. This adds to the robustness of the whole setup as any running agent can, at any time, discover other agents that are in communication range.
- Using the master_sync node topics of one ROS core are available to all the rest, i.e., every running SLAM agent can now subscribe to topics of the other agents that it is interested in (e.g., list of updated node positions).

5.2.2 map_merger_node

To facilitate in the overall inspection of the multi-robot graphSLAM procedure, we implemented the map_merger_node application.[6] Using similar operations as a multi-robot SLAM agent, the map_merger_node detects all running SLAM agents in its communication range and subscribes to their grid maps. When the operator is interested in getting an update on the multi-robot SLAM operation, map_merger_node fetches all possible grid maps and robot trajectories (with regards to the agent's respective frame of reference) and using the grid map aligning scheme presented in sec. 5.1, aggregates all the maps in a final version of the environment. Furthermore it transforms the corresponding trajectories according to the grid map alignment operation so that all the robot trajectories are displayed with regards to a common frame of reference, along with the aggregate map. Examples of running it, with 2 and 3 robots respectively are given in figs. 5.2.1,5.2.2.


csl_desktop_11312_119
1st robot map, trajectory
csl_desktop_119
2nd robot map, trajectory
Fused map
Combined map, trajectories
0.00 m/s
-1.00
0.00 rad/s

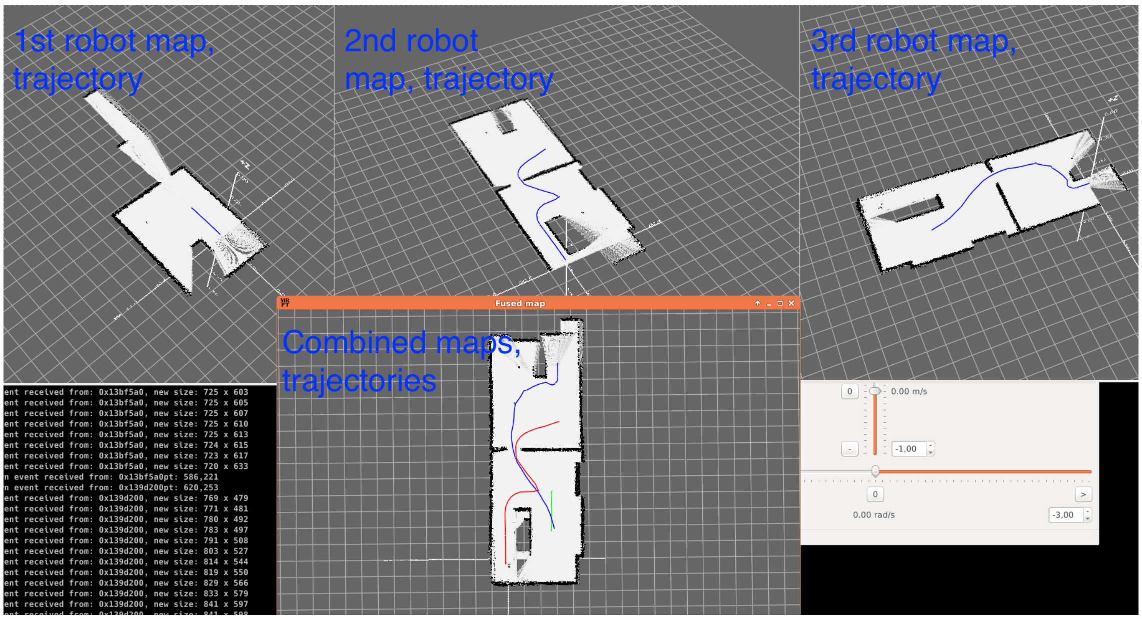
```
12697  3.8041e-05
09386 0.000218965
18965 3.91619e-05
12
y pose: (11.464,-2.883,-178.04deg)
ing covariance matrix:
0662556 -9.72869e-05
0098053   0.00103287
0103287  0.000133219
--

13:46:26.520606] Adding map to the fuse
d: 0x139dff0 => 1


13:46:26.521429] Adding #36 lines...
13:46:26.521458] Adding #41 lines...
eceived from: 0x139d200pt: 394,264
eceived from: 0x139d200pt: 590,307
eceived from: 0x139d200pt: 615,78
```

Figure 5.2.1: map_merger_node application - 2 robots, opposite direction, different rooms

[6]Application code is available as part of the mrpt_graphslam_2d package[7].


1st robot map, trajectory
2nd robot map, trajectory
3rd robot map, trajectory
Fused map
Combined maps, trajectories
0.00 m/s
-1,00
0.00 rad/s
-3,00

```
ent received from: 0x13bf5a0, new size: 725 x 603
ent received from: 0x13bf5a0, new size: 725 x 605
ent received from: 0x13bf5a0, new size: 725 x 607
ent received from: 0x13bf5a0, new size: 725 x 610
ent received from: 0x13bf5a0, new size: 725 x 613
ent received from: 0x13bf5a0, new size: 724 x 615
ent received from: 0x13bf5a0, new size: 723 x 617
ent received from: 0x13bf5a0, new size: 720 x 633
n event received from: 0x13bf5a0pt: 586,221
n event received from: 0x139d200pt: 620,253
ent received from: 0x139d200, new size: 769 x 479
ent received from: 0x139d200, new size: 771 x 481
ent received from: 0x139d200, new size: 780 x 492
ent received from: 0x139d200, new size: 783 x 497
ent received from: 0x139d200, new size: 791 x 508
ent received from: 0x139d200, new size: 803 x 527
ent received from: 0x139d200, new size: 814 x 544
ent received from: 0x139d200, new size: 819 x 550
ent received from: 0x139d200, new size: 829 x 566
ent received from: 0x139d200, new size: 833 x 579
ent received from: 0x139d200, new size: 841 x 597
```

Figure 5.2.2: map_merger_node application - 3 robots

5.3 Simulation

5.3.1 Simulation setup

To verify the behavior of the multi-robot algorithm we setup a simulated world consisting of two rooms (see fig. 5.3.1) along with an ample number of obstacles, and executed graphSLAM for a varying number of agents. The simulation environment were setup in `Gazebo`, along with the simulation of the vehicle dynamics, odometry, laser scans and camera measurements.

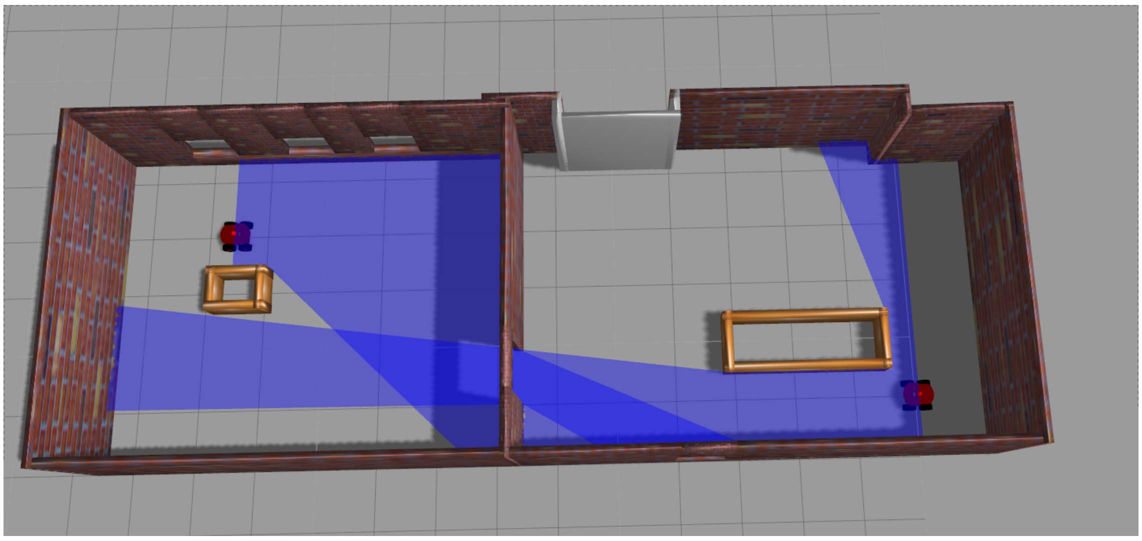


Figure 5.3.1: Environment used during the simulations in Gazebo

To drive the robots around the environment, we use teleoperation with velocity commands either using a keyboard/joystick or by recording a series of them and then playing them back. [8]

An image of the overall control panel during simulations is given below:

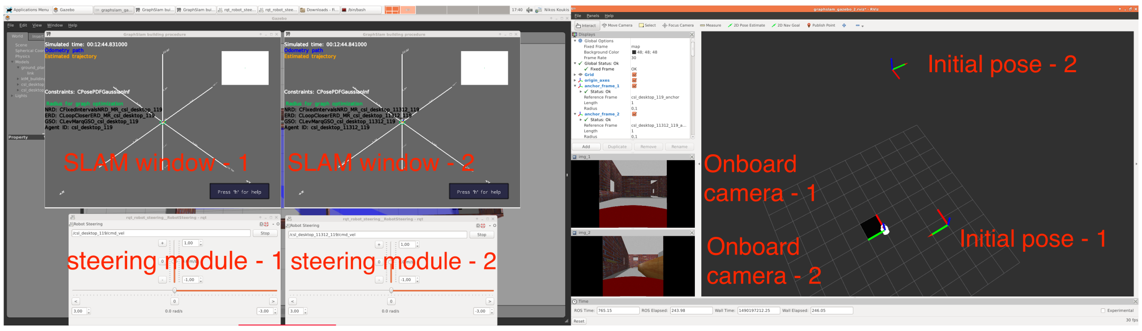


Figure 5.3.2: Control panel during simulations. Note that the number of corresponding windows spawned (SLAM window, steering module) depends exclusively on the number of agents the users has set up to run that is one SLAM window and one steering module pointing to the correct velocity topic is going to be spawned automatically.

To make the simulation as similar to the real-time experiments as possible each graph-SLAM agent was launched in a separate roscore. Along with it, a `master_discovery` and a `master_sync` node were launched to make the agents' communication possible.

For all the simulated agents the used deciders/optimizer along with some of their important configuration parameters are presented. For a full list of configuration parameters consult the corresponding .ini configuration files in `mrpt_graphslam_2d/config`.

Node Registration Decider: `CFixedIntervalsNRD_MR`

- **Registration distance:** $0.4m$
- **Registration angle:** 15 deg

Edge Registration Decider: `CLoopCloserERD_MR`

- **Distance to search for graph edges:** $0.5m$
- **Use scan matching for latest node:** YES
- **Num of previous nodes for scan-matching:** 10

graphSLAM Optimizer: `CLevMarqGSO`

- **Optimization distance:** $0.4m$

Note: As a naming convention, all the deciders/optimizers suffixed with _MR are multi-robot implementation of their corresponding single-robot counterpart.

To successfully configure and run a simulation in Gazebo we take the following steps:

1. Setup the environment of the computer that the simulations are going to be executed at. A configuration file should include directives such as:
   - Number of robot agents to be spawned

[8] Last method may not workas the velocity commands playback does not guarantee that the robots would traverse the exact trajectory that they did the time the commands were actually recorded. This is most likely due to the stochastic nature of the simulated noise of the simulator that we are using.

- Initial position of the latter
- Gazebo world to be used
- NRD/ERD/GSO classes for each one of the agents

An example of such configuration file is provided here[9]

```
source <path_to_configuration_file>
```

2. Setup the simulation environment, robot agents, as well as their corresponding sensors:

```
roslaunch csl_robots_gazebo setup_simulation_env.launch
```

3. On a second terminal, run single- / multi-robot graphSLAM according to *the environment variables of the current shell*:

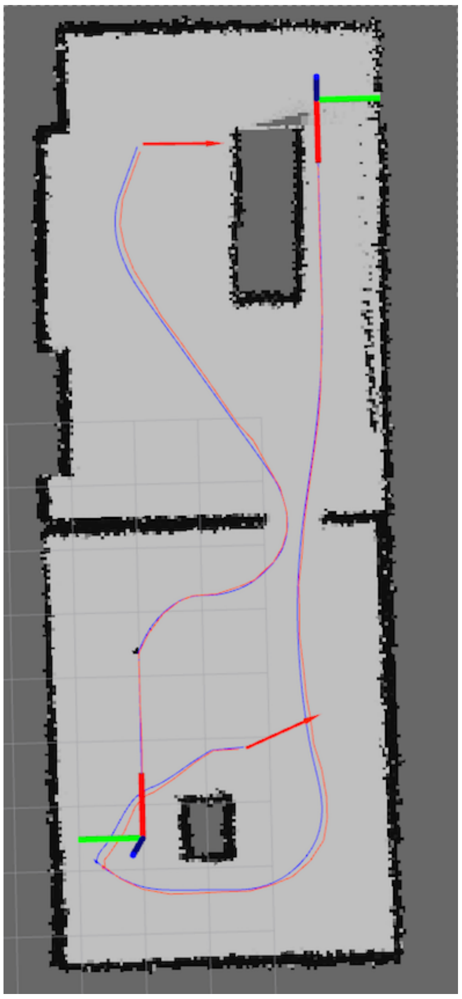
```
rosrun csl_robots_gazebo graphslam_launcher.py
```

5.3.2 Simulation results

The following is an analysis of the results produced during simulations. With regards to the map-matching scheme employed, the following remarks should also be made:

- As the original authors have suggested, during the map-matching procedure, a combination of the *Kanade-Lucas-Tomasi (KLT)* detector as well as the *lin-polar circular patches* descriptor is used.
- As previously explained in sec. 4, each robot caches the nodes and laser scans of its neighbors and when a minimum number of nodes and laser scans has been received, robot attempts to find a valid transform for own and neighbor's map. However, this strategy doesn't account for situations that robots may be in communication range, but there is no overlap in their maps. Thus, to make the mapping procedure as robust as possible, application users can impose that a prior estimate of that transform be available prior to map-matching. The latter can be estimated either in situations where the initial robot positions are available (trivial scenario) or by utilizing rendezvous of the agents (e.g., using a camera and an Aruco marker on each one of the agents). In the simulations presented, having set a minimum of 25 nodes was an adequate condition for the map-matching scheme to produce accurate results.

[9]`https://github.com/bergercookie/csl_mr_slam/blob/master/csl_robots_gazebo/config/slam/simulation_config.sh`


(a) Final occupancy grid map of 1st agent

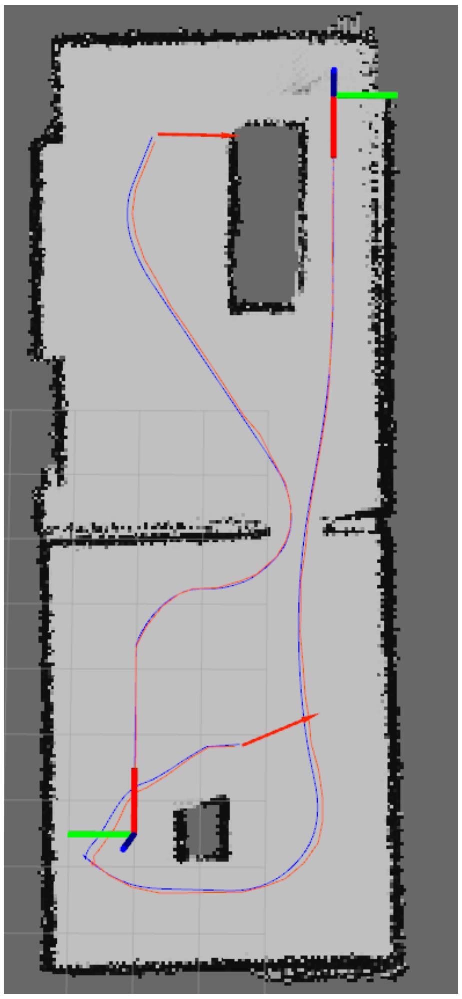
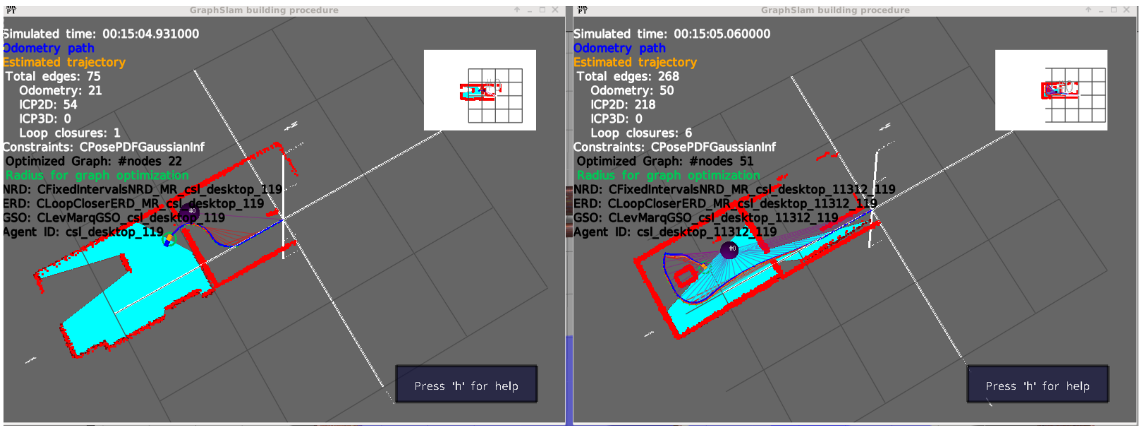

(b) Final occupancy grid map of 2nd agent

Figure 5.3.3: Final occupancy grid maps of both graphSLAM agents after the condensed measurements constraints are added for each agent

Figs. 5.3.4(a), 5.3.4(b) show the situation prior and after the map-matching algorithm and the computation of the constraint between the robot's own origin and its neighbor's origin. We notice that after the computation of the inter-graph transformation, the integration of the received nodes becomes a trivial task and the end result is successful.

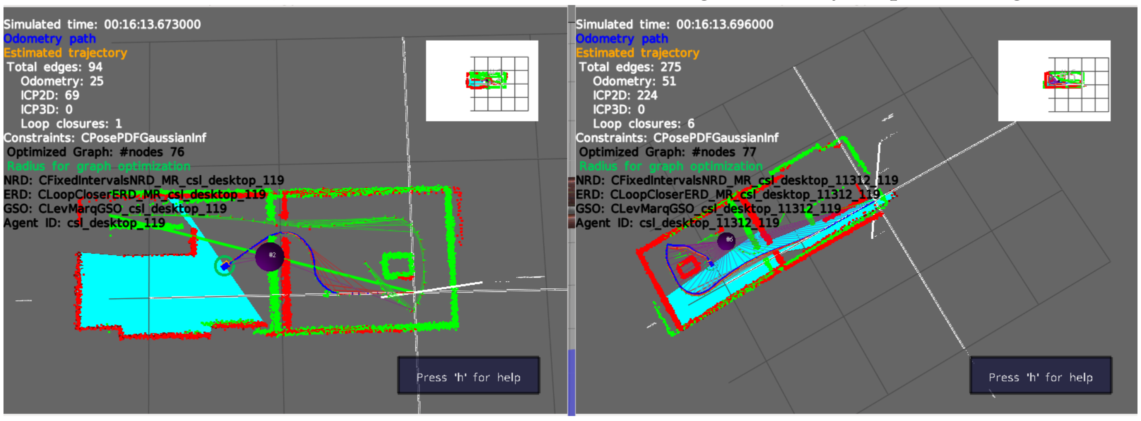
As mentioned in sec. 4 after the transformation is found, rest of the received nodes are integrated in batches (during simulations batch size was 5 nodes.) so that the addition of scan-matching edges and optimization of the condensed measurements graph is done on the transmitter's side. Final map and trajectory of each agent is depicted in 5.3.3(a), 5.3.3(b).



```
Simulated time: 00:15:04.931000
Odometry path
Estimated trajectory
Total edges: 75
   Odometry: 21
   ICP2D: 54
   ICP3D: 0
   Loop closures: 1
Constraints: CPosePDFGaussianInf
Optimized Graph: #nodes 22
Radius for graph optimization
NRD: CFixedIntervalsNRD_MR_csl_desktop_119
ERD: CLoopCloserERD_MR_csl_desktop_119
GSO: CLevMarqGSO_csl_desktop_119
Agent ID: csl_desktop_119
```

```
Simulated time: 00:15:05.060000
Odometry path
Estimated trajectory
Total edges: 268
   Odometry: 50
   ICP2D: 218
   ICP3D: 0
   Loop closures: 6
Constraints: CPosePDFGaussianInf
Optimized Graph: #nodes 51
Radius for graph optimization
NRD: CFixedIntervalsNRD_MR_csl_desktop_11312_119
ERD: CLoopCloserERD_MR_csl_desktop_11312_119
GSO: CLevMarqGSO_csl_desktop_11312_119
Agent ID: csl_desktop_11312_119
```

(a) Situation **prior** to condensed measurements registration by map-matching



```
Simulated time: 00:16:13.673000
Odometry path
Estimated trajectory
Total edges: 94
   Odometry: 25
   ICP2D: 69
   ICP3D: 0
   Loop closures: 1
Constraints: CPosePDFGaussianInf
Optimized Graph: #nodes 76
Radius for graph optimization
NRD: CFixedIntervalsNRD_MR_csl_desktop_119
ERD: CLoopCloserERD_MR_csl_desktop_119
GSO: CLevMarqGSO_csl_desktop_119
Agent ID: csl_desktop_119
```

```
Simulated time: 00:16:13.696000
Odometry path
Estimated trajectory
Total edges: 275
   Odometry: 51
   ICP2D: 224
   ICP3D: 0
   Loop closures: 6
Constraints: CPosePDFGaussianInf
Optimized Graph: #nodes 77
Radius for graph optimization
NRD: CFixedIntervalsNRD_MR_csl_desktop_11312_119
ERD: CLoopCloserERD_MR_csl_desktop_11312_119
GSO: CLevMarqGSO_csl_desktop_11312_119
Agent ID: csl_desktop_11312_119
```

(b) Situation **after** condensed measurements registration by map-matching. Thick green line indicates the inter-graph constraint as computed by the grid map aligning algorithm. Green subgraph is the condensed measurements graph of the nodes received by the other agent. Green points map is a recreated version of the neighbors' map based only on the received incorporated nodes and their corresponding measurements.

Figure 5.3.4: Situation prior and after first incorporation of neighbor's nodes

A demo video of the simulation using two agents is provided here[10].

5.4 Experimental Results

As a final step in the development of the multi-robot graphSLAM algorithm, we tested it in a real-time environment, much similar to the single-robot experiments (sec. 3.6).

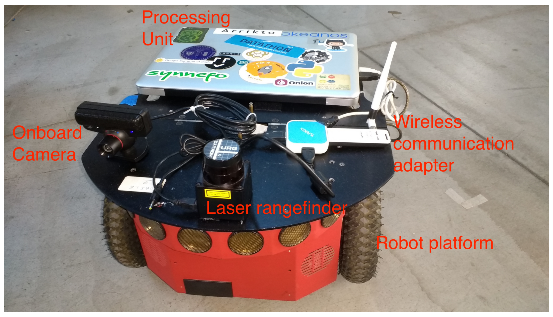
Similar to the simulation, purpose of the experiments was to verify that the communication strategy also worked real-time and the robots would successfully exchange and integrate their counterparts measurements in their own map. To do that and to simplify the overall operation, we use two robots (namely `odroidxu3` and `nickkoukubuntu` from the corresponding computer hostname) initially situated in two different and adjacent rooms. These are to exchange graph information and measurements. One of them (odroidxu3) traverses both rooms while nickkoukubuntu makes a short right turn in its own room. After their corresponding movements and each having cached the nodes and measurements sent over the network by the other agent, they execute map-matching and integrate

[10] `https://www.youtube.com/watch?v=4RKS2jrvsYE`

the fetched nodes and measurements in their own map and graph. The successful integration can be verified from the fact that even though nickkoukubuntu doesn't traverse the second room, it successfully creates a map of it from the information received from odroidxu3. On the other hand, odroidxu3 rectifies the part of the environment that both robots have traversed and mapped.

In fig. 5.4.1(a) the two robot agents used during multi-robot graphSLAM are depicted.

(a) Pioneer 2dx models along with their processing units and data acquisition devices, as used in the multi-robot experiments.



(b) Components of a graphSLAM agent along with their corresponding explanation. We should remark that the onboard camera was used just as reference of the surrounding environment of the robot (information not utilised in the graphSLAM algorithm)

Figure 5.4.1: Agents of multi-robot experiment

5.4.1 Configuring an agent for online multi-robot SLAM

Configuration of a robot that is to run in a multi-robot graphSLAM framework is identical to the single-robot case with the only difference being in the decider/optimizer classes that are to be used. For more on this, see sec. 3.5.1.

5.4.2 Network Setup

For the multi-robot SLAM to take place, a communications system must be setup between the agents. The requirements for such a system are as follows:

Reliability - Robustness to connection problems
It is assumed (as in real-time scenarios) that the communication links between the agents are not reliable and that the agents are not necessarily connected at every step during execution. For this to work the selected protocol must be *decentralized*, robust to network failures and broken communication links between arbitrary robotic agents. This crosses out the possibility of master-slaves models where one robot hands out the instructions and waits for feedback from the rest.

Easy to setup
For maximum efficiency and for the experimental setup to be established as fast as possible the communication protocol has to be simple and straightforward.

Easy to introspect
An external operator, or *Central Node*, of the system should be able to query and visualize both the individual agent trajectory and map as well as the aggregated

results. This would provide insight into the algorithm functionality as well as assist in the search for potential bugs of the algorithm.

Having laid out the basic prerequisites for the application we chose to implement the following features:

Ad-hoc network setup
A standard predetermined ad-hoc network which all running agents as well as the Central Node are going to use for communication.

DNS Server
To successfully exchange information over the ad-hoc network, agents should be able to resolve the hostnames of their peers, that is be able to determine their IP address based on their hostname. Thus a Domain Name System (DNS) service is to be set up. We chose to setup the latter the Central Node of the operation.

World Wide Web access of agents [OPTIONAL]
An optional, but valuable, step in the management and configuration of the robots (e.g., installation of precompiled packages, download of sources, troubleshooting) nodes of the ad-hoc network are granted internet access using appropriate firewall configuration in the Central node of the system (which is connected to the world by a lan cable). See the access_internet.conf.template[11], access_internet_adhoc_prov.sh[12] scripts for more on this.

Below, we provide instructions on how to reproduce such a setup.

Ad-Hoc Network

During the multi-robot experiments the following configuration was used:

Wireless Configuration	IP Configuration
Mode: Ad-Hoc	**IP-Addresses range:** 10.8.0.1-254
ESSID: multi_robot_exp	**Netmask:** 255.255.255.0
Encryption: Off	**Broadcast address:** 10.8.0.255
	DNS Nameserver: 10.8.0.1

To configure an agent (i.e., a computer mounted the robotic platform) to participate in the wireless Ad-Hoc network, see the setup_adhoc.py[13] script. The latter registers an upstart script on the node it is executed so that:

- Node at hand, automatically joins the network on startup with a predefined IP address.
- Connection to the ad-hoc persists even after reboots or after restarting the corresponding network interface (e.g., ejecting and reinserting the adapter)
- A DNS server entry is registered which can be quarried to resolve the names of the other running agents in the common network.

[11]`https://github.com/bergercookie/csl_mr_slam/blob/master/csl_hw_setup/scripts/ad_hoc_network/access_internet.conf.template`

[12]`https://github.com/bergercookie/csl_mr_slam/blob/master/csl_hw_setup/scripts/ad_hoc_network/access_internet.conf.template`

[13]`https://github.com/bergercookie/csl_mr_slam/tree/master/csl_hw_setup/scripts/ad_hoc_network/setup_adhoc.py`

For the multi-robot experiment at hand the following hostname, IP address pairs were utilised:

- csldesktop - 10.8.0.1 (central node)
- nickkoukubuntu - 10.8.0.16
- odroidxu3 - 10.8.0.17

DNS Server

A basic requirement to use a multimaster ROS configuration is that every agent (running a `roscore` instance) knows the IP address and the corresponding hostname of every other agent. This is handled by a DNS server running in one of the agents (or by multiple DNS servers for setups with more agents) that handles the DNS queries of every other agent in the network. To keep the level of complexity low, and since a basic setup is needed, dnsmasq[14] was used for this purpose. `dnsmasq` looks for the hostnames and corresponding IP addresses in the `/etc/hosts` file. It is strongly advised that the `localise-queries` flag is set in the dnsmasq configuration file (by default found in `/etc/dnsmasq.conf`), so that the DNS server hands out the in-ad-hoc addresses first. [15]

Setup the dnsmasq by issuing the following commands:
```
[sudo] apt-get install dnsmasq
[sudo] /etc/init.d/dnsmasq restart
```

Multimaster node configuration

By now every agent must have a unique IP address in the Ad-Hoc network which should also be registered in the DNS server's `/etc/hosts/` file.

We can verify this by pinging from one host to the other:

```
ping <another-address-in-ad-hoc> -I <ad-hoc-interface>
```

We can now use the multimaster_fkie ROS package for managing the individual `roscores` in a consistent manner. For a detailed manual on using the multimaster_fkie package see here[16].

Note: For the master_discovery script to be effective *IP forwarding* must be enabled on the host and the multicast traffic should also be routed through the wireless interface used in the Ad-Hoc network. This can be implemented by following the procedure below:

1. Modify/append `/etc/sysctl.conf` so that:
   ```
   net.ipv4.ip_forward=1
   net.ipv4.icmp_echo_ignore_broadcasts=0
   ```
2. Reload sysctl configuration
   ```
   $ sysctl -p
   ```

[14]`http://www.thekelleys.org.uk/dnsmasq/doc.html`
[15]Make sure that the other agents have set as a DNS server the host that dnsmasq runs on
[16]`http://www.iri.upc.edu/files/scidoc/1607-Multi-master-ROS-systems.pdf`

3. Route multicast traffic to the correct wireless interface: [17]
   `route add -net 224.0.0.0 netmask 224.0.0.0 <ad-hoc-interface>`

4. Ping the multicast address via the ad-hoc network interface to verify the correct behavior: `ping 224.0.0.1 -I <ad-hoc-interface>`

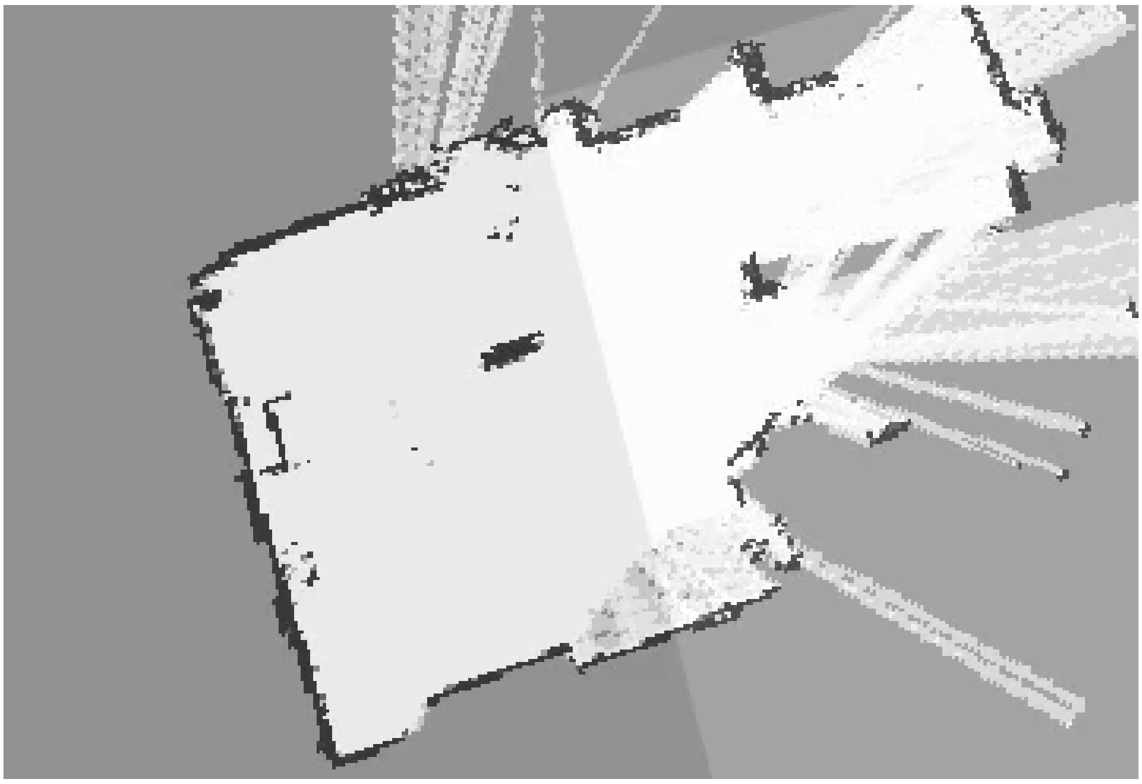
If the above procedure works correctly, then after launching the `master_discovery` and `master_sync` processes on each agent, topics of one agent should be also be visible and readable from another agent, that is:

```
agent1 $ rostopic pub /kalimera std_msgs/String -r 10 "kalimera!"
agent2 $ rostopic echo /kalimera # should print "data: kalimera!"
```

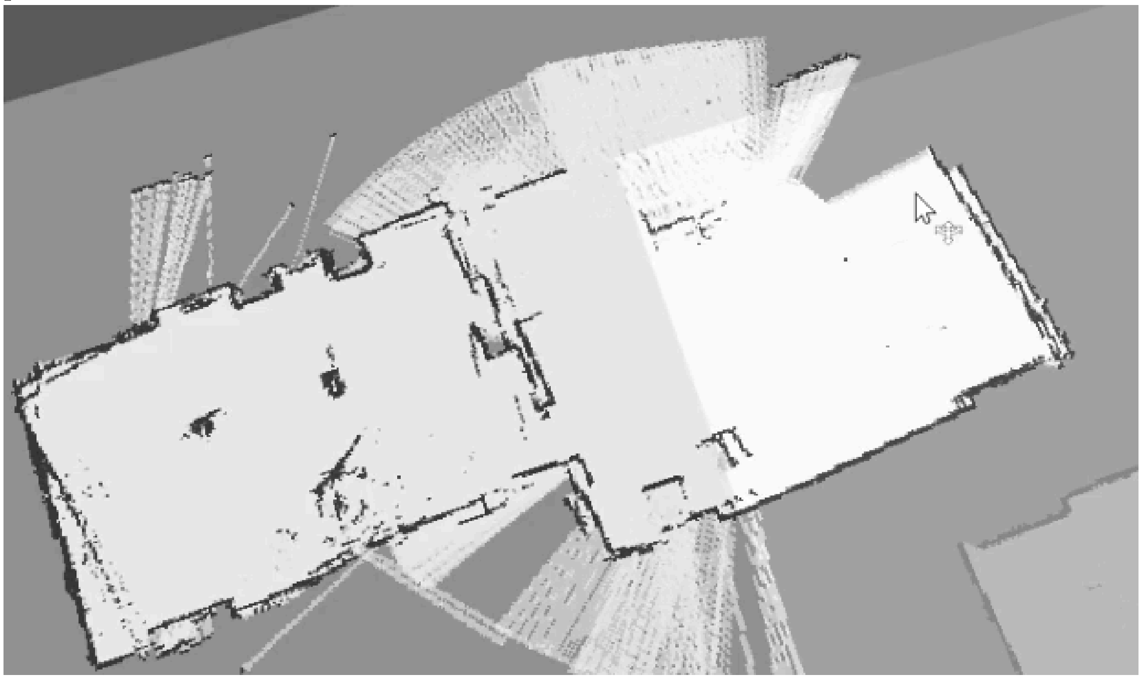
5.4.3 1st experiment - Conclusions

Results prior and after the integration of received nodes, for one of the agents, are given in fig. 5.4.2(a) and fig. 5.4.2(b). A video demonstration of the online multi-robot graphSLAM experiment is also provided here[18]

[17] This is taken care of by the `setup_ad_hoc.py` script
[18] `https://www.dropbox.com/s/zm2nj1jeprnsfaf/20170426_mr_graphslam_real_2.mp4?dl=0`



(a) Situation **prior** to the integration of the received measurements. Robot knows only about the part that itself has traversed.

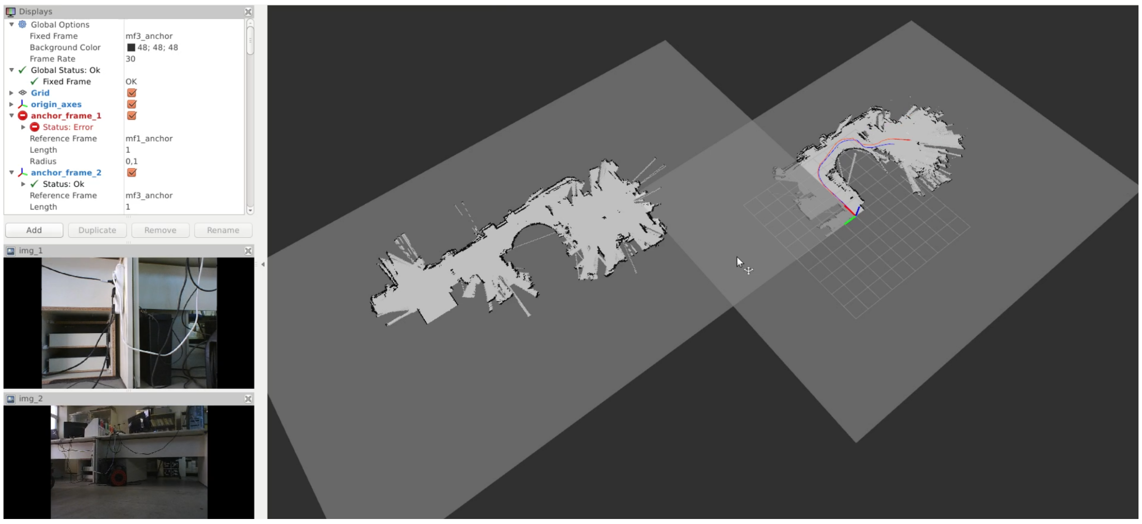


(b) Situation **after** the successful map-matching and integration of the received measurements. After this, robot is aware of the area traversed by its peer.

Figure 5.4.2: Successful map-matching and integration of the received measurements.

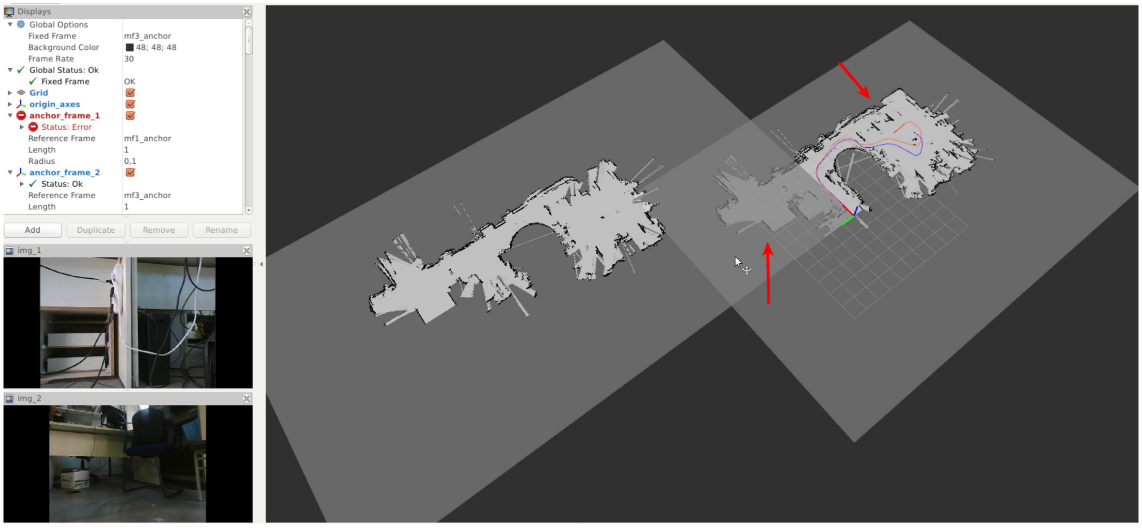
5.4.4 2nd experiment - Conclusions

A second experiment has been conducted in the basement of Ktirio M. The robots used are identical to the previous experiment. Results prior and after the registration of the neighbor's nodes are provided in fig. 5.4.3(a) and fig. 5.4.3(b). A video demonstration of the online multi-robot graphSLAM experiment is also provided here[19]



```
Global Options
    Fixed Frame          mf3_anchor
    Background Color     ■ 48; 48; 48
    Frame Rate           30
Global Status: Ok
    ✓ Fixed Frame        OK
Grid                     ✓
origin_axes              ✓
anchor_frame_1           ✓
    Status: Error
    Reference Frame      mf1_anchor
    Length               1
    Radius               0,1
anchor_frame_2           ✓
    ✓ Status: Ok
    Reference Frame      mf3_anchor
    Length               1
```

(a) Situation **prior** to the integration of the received measurements.



```
Global Options
    Fixed Frame          mf3_anchor
    Background Color     ■ 48; 48; 48
    Frame Rate           30
Global Status: Ok
    ✓ Fixed Frame        OK
Grid                     ✓
origin_axes              ✓
anchor_frame_1           ✓
    Status: Error
    Reference Frame      mf1_anchor
    Length               1
    Radius               0,1
anchor_frame_2           ✓
    ✓ Status: Ok
    Reference Frame      mf3_anchor
    Length               1
```

(b) Situation **after** the successful map-matching and integration of the received measurements. After this, robot is aware of the area traversed by its peer.

Figure 5.4.3: Successful map-matching and integration of the received measurements. Experiment was executed around the pool of the underwater lab at Ktirio M.

[19]`https://www.dropbox.com/s/xx1mztrtddkm4ky/real_mr2.mp4?dl=0`

Chapter 6

Conclusions - Future Directions

In the current section we sum up the work that we have conducted in the master thesis, and we set the goals for future works/projects that are to be implemented on top of it.

As part of this master thesis we have:

- Conducted a thorough analysis of the dominant strategies, problems and current trends in the field of single as well as multi-robot SLAM, specializing our research in graph-based strategies, which show increased interest and potential at this stage.
- Implemented a framework for conducting single-robot graphSLAM using either prerecorded datasets or real-time data. Code has been accepted and is publicly available to use, as part of the popular open-source MRPT[1] robotics toolkit. This includes the implementation of a robust loop closing scheme initially devised and published by Edwin Olson.
- Rigorously tested the single-robot algorithm both in simulations and in real-time environments using a 4-wheeled Pioneer as well as a youbot platform.
- Implemented a multi-robot technique for robot agents to share part of their graph so that they assist in their neighbor's SLAM procedure. Strategy was tested in simulations specifically designed in the Gazebo environment as well as real-time online experiments. Implementation includes the communication of the agents via a decentralized ad-hoc network and the utilization of the *multicast* protocol so that each agent efficiently and robustly finds its neighbors in the common ad-hoc network.
- Utilized a map-matching algorithm for finding a valid transform between own grid map and the grid map reconstructed from a neighbor's fetched measurements and nodes. Multi-robot related algorithms are publicly available in ros.org/mrpt_graphslam_2d[2].

Based on this work, we suggest the following modifications/extensions to improve the algorithm features, usability and overall efficiency:

Integration of 3rd party optimization framework
Modern optimization frameworks such as g2o[3] or iSAM[4] can exploit the sparse and incremental nature of the SLAM problem. Thus integration of either of those

[1]`http://www.mrpt.org`
[2]`http://wiki.ros.org/mrpt_graphslam_2d`
[3]`https://github.com/RainerKuemmerle/g2o`
[4]`http://people.csail.mit.edu/kaess/isam/`

schemes as a GraphSLAM Optimizer class in the mrpt-graphslam library is considered a major improvement. This will improve the overall efficiency in the optimization procedure, as currently only a non-linear optimization framework based on the Levenberg-Marquardt algorithm is available.

As of March 2017, the *SE-Sync* optimization algorithm [67] is also available as the first *certifiably correct, global optimizer* for a system of non-linear constraints. Since the vast majority of optimization modules so far, act on non-convex domains and thus, locally optimize the given set of constraints, SE-Sync sets itself apart as the first algorithm to act on a global scale by utilizing Special Euclidean Groups and Riemannian manifolds.

Adaptive node registration decider
Every node registration decider class implemented thus far, adds new nodes in the graph in fixed distance and angle intervals, using either odometry and/or laser scan measurements. It would be useful especially in large areas mapping scenarios to consider adding nodes only when new information is available to the robot, thus reduce the overall complexity of the problem

Node reduction scheme
Putting an upper bound in the maximum number of nodes that can be registered in a graph can potentially apply a bound in the computational complexity, storage requirements and overall required resources of the entire algorithm. Notable works in this field are presented in [68], [58], [69], [39].

Support for visual sensors
Current implementations of the `mrpt-graphslam` library can utilize odometry as well as laser scan measurements. It would be valuable to extend the support of available sensors such as 2D cameras, Kinect cameras either for 2D or 3D mapping.

Support for 3D mapping
Even though the main `CGraphSlamEngine` class already supports the construction and management of 3D graphs as well, an edge registration decider that does so hasn't been implemented yet. This would comprise a significant step towards extending the library support either for utilizatizing 3D measurements in 2D mapping scenarios or directly using it in full 3D SLAM cases.

Support for active exploration
Active SLAM is a variant in which the robot itself navigates to places still unknown its map to explore as much of its surrounding area as possible.

Additional suggested improvements in the algorithm are presented here[5].

[5]`https://github.com/MRPT/mrpt/milestone/9`

Bibliography

[1] E. A. Wan and R. Van Der Merwe, ``The unscented Kalman filter for nonlinear estimation,'' *Technology*, vol. v, pp. 153--158, 2000.

[2] K. Madsen, H. B. Nielsen, and O. Tingleff, ``IMM METHODS FOR NON-LINEAR LEAST SQUARES PROBLEMS,'' 2004.

[3] S. Saeedi, M. Trentini, M. Seto, and H. Li, ``Multiple-Robot Simultaneous Localization and Mapping: A Review,'' *Journal of Field Robotics*, 2016.

[4] F. Lu and E. Milios, ``Robot pose estimation in unknown environments by matching 2D range scans,'' *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 249--275, 1994.

[5] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. Van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, ``Stanley: The Robot that Won the DARPA Grand Challenge,'' *Journal of Field Robotics*, vol. 23, no. 9, pp. 661--692, 2006.

[6] J. A. Castellanos, G. Grisetti, and M. Lazaro, ``Multi-Robot SLAM using Condensed Measurements,'' pp. 1069--1076, 2013.

[7] S. Saeedi, L. Paull, M. Trentini, and H. Li, ``Multiple robot simultaneous localization and mapping,'' *IEEE International Conference on Intelligent Robots and Systems*, pp. 853--858, 2011.

[8] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller, ``An Atlas framework for scalable mapping,'' *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, no. September, pp. 1899--1906, 2003.

[9] G. Erinc and S. Carpin, ``Anytime merging of appearance based maps,'' in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1656--1662, IEEE, may 2012.

[10] R. C. Smith and P. Cheeseman, ``On the Representation and Estimation of Spatial Uncertainty,'' *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56--68, 1986.

[11] H. Durrant-Whyte and T. Bailey, ``Simultaneous localization and mapping: Part I,'' *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99--108, 2006.

[12] J. Sola, ``Simulataneous localization and mapping with the extended Kalman filter,'' *unpublished. Available: http://www. joansola. eu/JoanSola/eng/JoanSola. html*, pp. 1--35, 2013.

[13] S. J. Julier and J. K. Uhlmann, ``A counter example to the theory of simultaneous localization and map building,'' in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 4, pp. 4238--4243, IEEE, 2001.

[14] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, ``Simultaneous Localization and Mapping with Sparse Extended Information Filters,'' *The International Journal of Robotics Research*, vol. 23, no. 7, pp. 693--716, 2003.

[15] E. B. Olson, ``Robust and Efficient Robotic Mapping,'' *Work*, vol. 31, pp. 265--272, 2008.

[16] M. Walter, R. Eustice, and J. Leonard, ``A Provably Consistent Method for Imposing Sparsity in Feature-Based SLAM Information Filters,'' in *Robotics Research*, pp. 214--234, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[17] R. Eustice, M. Walter, and J. Leonard, ``Sparse extended information filters: insights into sparsification,'' in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3281--3288, IEEE, 2005.

[18] J. E. Guivant and E. M. Nebot, ``Optimization of the simultaneous localization and map-building algorithm for real-time implementation,'' *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242--257, 2001.

[19] J. Guivant and E. Nebot, ``Solving Computational and Memory Requirements of Feature Based Simultaneous Localization and Map Building Algorithms,'' no. Cml, 2002.

[20] S. J. Julier and J. K. Uhlmann, ``A New Extension of the Kalman Filter to Nonlinear Systems,''

[21] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, ``FastSLAM: A factored solution to the simultaneous localization and mapping problem,'' *Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence*, vol. 68, no. 2, pp. 593--598, 2002.

[22] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit, ``FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges,'' *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1151--1156, 2003.

[23] G. Grisetti, C. Stachniss, and W. Burgard, ``Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling,'' *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2005, pp. 2432--2437, 2005.

[24] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, ``Fast and accurate SLAM with Rao-Blackwellized particle filters,'' *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30--38, 2007.

[25] A. Chatterjee and Amitava, ``Differential evolution tuned fuzzy supervisor adapted extended Kalman filtering for SLAM problems in mobile robots,'' *Robotica*, vol. 27, p. 411, may 2009.

[26] M. J. Milford and G. F. Wyeth, ``Mapping a suburb with a single camera using a biologically inspired SLAM system,'' *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1038--1053, 2008.

[27] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, and M. Milford, ``OpenRatSLAM: An open source brain-based SLAM system,'' *Autonomous Robots*, vol. 34, no. 3, pp. 149--176, 2013.

[28] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison, ``SLAM++: Simultaneous localisation and mapping at the level of objects,'' in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1352--1359, IEEE, jun 2013.

[29] H. Choset and K. Nagatani, ``Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization,'' *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125--137, 2001.

[30] J. O. Wallgrun, ``Voronoi Graph Matching for Robot Localization and Mapping,'' *Transactions on Computational Science Ix*, vol. 6290, pp. 76--108, 2010.

[31] J. Boal, Á. Sánchez-Miralles, and Á. Arranz, ``Topological simultaneous localization and mapping: a survey,'' *Robotica, Cambridge University Press*, vol. 32, no. 5, pp. 803--821, 2014.

[32] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, ``Multiple relative pose graphs for robust cooperative mapping,'' *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3185--3192, 2010.

[33] F. Dellaert and M. Kaess, ``Square Root SAM,'' *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99--108, 2006.

[34] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, ``A tutorial on graph-based SLAM,'' *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31--43, 2010.

[35] E. M. F. Lu, ``Globally Consisten Range Scan Alignment for Environment Mapping,'' *Statewide Agricultural Land Use Baseline 2015*, vol. 1, 1997.

[36] J.-S. Gutmann and K. Konolige, ``Incremental mapping of large cyclic environments,'' *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No.99EX375)*, pp. 318--325, 1999.

[37] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, ``iSAM2: Incremental smoothing and mapping using the Bayes tree,'' *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216--235, 2012.

[38] S. Thrun and M. Montemerlo, ``The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures,'' *Robotics Research, The International Journal of*, vol. 25, no. 5-6, pp. 403--429, 2006.

[39] C. Estrada, J. Neira, and J. D. Tardós, ``Hierarchical SLAM: Real-time accurate mapping of large environments,'' *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588--596, 2005.

[40] E. Olson, ``Recognizing places using spectrally clustered local matches,'' *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1157--1172, 2009.

[41] A. Nuchter, K. Lingemann, J. Hertzberg, and H. Surmann, ``6D SLAM with approximate data association,'' in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pp. 242--249, IEEE.

[42] C. Stachniss, ``Least Squares Approach to SLAM,'' tech. rep.

[43] a. Howard, ``Multi-robot Simultaneous Localization and Mapping using Particle Filters,'' *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1243--1256, 2006.

[44] A. Birk and S. Carpin, ``Merging occupancy grid maps from multiple robots,'' *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384--1397, 2006.

[45] L. Carlone, M. K. Ng, J. Du, B. Bona, M. Indri, L. Carlone, M. K. Ng, J. Du, ·. B. Bona, ·. M. Indri, B. Bona, M. Indri, M. Kaouk Ng, J. Du, B. Bona, and M. Indri, ``Simultaneous Localization and Mapping Using Rao-Blackwellized Particle Filters in Multi Robot Systems,'' *Journal of Intelligent & Robotic Systems*, vol. 63, no. 2, pp. 283--307, 2010.

[46] L. A. A. Andersson and J. Nygårds, ``C-SAM: Multi-robot SLAM using square root information smoothing,'' *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2798--2805, 2008.

[47] N. Michael, S. Shen, K. Mohta, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro, ``Collaborative mapping of an earthquake damaged building via ground and aerial robots,'' in *Springer Tracts in Advanced Robotics*, vol. 92, pp. 33--47, Springer Berlin Heidelberg, 2014.

[48] Http://chrony.tuxfamily.org/, ``chrony – Introduction.''

[49] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte, ``Towards multi-vehicle simultaneous localisation and mapping,'' *Proceedings of the IEEE International Conference on Robotics and Automation*, no. May 2002, pp. 2743--2748, 2002.

[50] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte, ``An efficient approach to the simultaneous localisation and mapping problem,'' in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 406--411, IEEE, 2002.

[51] S. B. Williams, ``Efficient Solutions to Autonomous Mapping and Navigation Problems,'' *System*, no. September, 2001.

[52] S. Thrun and Y. Liu, ``Multi-robot SLAM with Sparse Extended Information Filers,'' *The International Journal of Robotics Research*, vol. 15, pp. 254--266, 2005.

[53] E. Nettleton, P. Gibbens, and H. Durrant-Whyte, ``Closed form solutions to the multiple-platform simultaneous localization and map building (SLAM) problem,'' in *Proceedings of SPIE* (B. V. Dasarathy, ed.), vol. 4051, p. 428, International Society for Optics and Photonics, apr 2000.

[54] X. S. Zhou and S. I. Roumeliotis, ``Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case,'' *IEEE International Conference on Intelligent Robots and Systems*, pp. 1785--1792, 2006.

[55] P. Besl and N. D. McKay, ``A method for registration of 3-D shapes,'' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239--256, feb 1992.

[56] S. Rusinkiewicz and M. Levoy, ``Efficient Variants of the ICP Algorithm,''

[57] O. Sorkine-Hornung and M. Rabinovich, ``Least-Squares Rigid Motion Using SVD,'' 2017.

[58] J. L. Blanco, J. Gonzalez, and J. A. Fern??ndez-Madrigal, ``Consistent observation grouping for generating metric-topological maps that improves robot localization,'' *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2006, no. May, pp. 818--823, 2006.

[59] K. Levenberg, ``A Method for the solution of certain non-linear probles in least squares.,'' *Q. Appl. Math*, vol. 11, no. 2, pp. 164--168, 1944.

[60] D. W. Marquardt, ``An Algorithm for Least-Squares Estimation of Nonlinear Parameters,'' *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, pp. 431--441, jun 1963.

[61] M. I. a. Lourakis, ``A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar,'' *Matrix*, vol. 3, p. 2, 2005.

[62] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, ``On measuring the accuracy of SLAM algorithms,'' *Autonomous Robots*, vol. 27, no. 4, pp. 387--407, 2009.

[63] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardós, ``A Comparison of SLAM Algorithms Based on a Graph of Relations,'' *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pp. 2089--2095, 2009.

[64] A. Tsiamis, J. Tumova, C. P. Bechlioulis, G. C. Karras, D. V. Dimarogonas, and K. J. Kyriakopoulos, ``Decentralized leader-follower control under high level goals without explicit communication,'' *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 5790--5795, 2015.

[65] a. Howard, M. J. Mataric, and G. S. Sukhatme, ``Putting the'I'in'Team': An egocentric approach to cooperative localization,'' *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1, no. May, pp. 868--874, 2003.

[66] J.-l. Blanco, J. Gonz, and J.-a. Fern, ``A Robust , Multi-Hypothesis Approach to Matching Occupancy Grid Maps,'' vol. 31, pp. 687--701, 2013.

[67] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, ``SE-Sync: A Certifiably Correct Algorithm for Synchronization over the Special Euclidean Group *,'' 2017.

[68] H. Kretzschmar and C. Stachniss, ``Information-theoretic compression of pose graphs for laser-based {SLAM},'' *International Journal of Robotics Research*, vol. 31, no. 11, pp. 1219--1230, 2012.

[69] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, ``Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping,''

Appendix A

SLAM in Bayesian representation

Current section presents the formulation of SLAM in the Bayesian context. This is the first step in the mathematical formulation of strategies such as the EKF or the PF. Section at hand is heavily based on [11].

Consider a mobile robot moving through its environment. Taking relative observations of unknown landmark using onboard sensors, as illustrated in fig. A.0.1 At time instance k the following quantities are defined:

$\boldsymbol{x}_k$ The state vector describing the estimated pose (location and orientation) of the robot.

$\boldsymbol{u}_k$ The control vector *applied at time* $k-1$ to drive the vehicle to a state $\boldsymbol{x}_k$ at time k.

$\boldsymbol{m}_i$ A vector describing *the location* of the i^{th} landmark. Its location is assumed to be time invariant.

$\boldsymbol{z}_{ik}$ An observation of the i^{th} landmark at timestep k. In case there are multiple landmark observations bundled together at a specific timestep (as is often the case), or when the specific landmark is not relevant, the i index is dropped, thus z_k.

We also define the following vectors:

The history of vehicle poses
$\boldsymbol{X}_{0:k} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_k\} = \{\boldsymbol{X}_{0:k-1}, \boldsymbol{x}_k\}$

The history of control inputs
$\boldsymbol{U}_{0:k} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\} = \{\boldsymbol{U}_{0:k-1}, \boldsymbol{u}_k\}$

The set of all landmarks
$\boldsymbol{m} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots \boldsymbol{m}_n\}$

The set of all landmark observations
$\boldsymbol{Z}_{0:k} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots \boldsymbol{z}_k\} = \{\boldsymbol{Z}_{0:k-1}, \boldsymbol{z}_k\}$

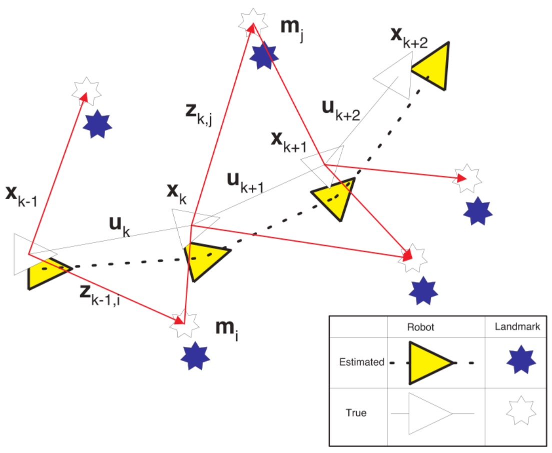


Figure A.0.1: The SLAM problem in Bayesian representation. The robot positions and landmark positions as estimated by the SLAM algorithm are given in yellow blue color respectively. The ground-truth (real) robot and landmark positions are given as the transparent icons. Notice that the latter cannot be known or directly measured at any stage of the algorithm. Sensor information (observations) provide associate (provide an estimation between) a robot and landmark position.

Having defined the corresponding notation, we move on to the Bayesian formulation. In mathematical terms, SLAM problems come down to computing the following probability distribution for all times k:

$$P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)$$

The latter describes the *joint posterior density* of the landmark locations and latest robot estimated pose (at time k) based on all recorded observations, and control inputs up to time k as well as the assumption of the initial robot position $\boldsymbol{x}_0$. Notice that we are looking to derive a recursive scheme, that is, starting with an estimate of the distribution $P(\boldsymbol{x}_{k-1}, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k-1})$ at time $k-1$ and using appropriate functions to model the dynamics of movement and observation of the robot and sensors, we want to compute $P(\boldsymbol{x}_k, \boldsymbol{m}|\cdot)$.

Observation model

Describes the probability of a measurement, assuming that the landmarks (map) as well as the origin of measurement (robot position) are known:

$$P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{A.0.1}$$

Motion model

Assuming that the robot movement is a Markov process, the probability distribution of the robot position at timestep k depends only on the pose at the previous timestep $k-1$ as well as the current control input $\boldsymbol{u}_k$:

$$P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{A.0.2}$$

The SLAM algorithm thus comprises on a two-step recursion scheme based on a prediction step (or time update) and a correction step (measurement update) which are given by the following equations:

Time update

$$P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) = \int P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \times P(\boldsymbol{x}_{k-1}, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0,k-1}, \boldsymbol{x}_0) d\boldsymbol{x}_{k-1} \tag{A.0.3}$$

Measurement update

$$P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) = \frac{P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}) P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)}{P(\boldsymbol{z}_k|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k}} \tag{A.0.4}$$

Thus, using equations eqn. A.0.3,and eqn. A.0.4 we can compute the joint posterior $P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{x}_0)$ for the robot state $\boldsymbol{x}_k$ and the map $\boldsymbol{m}$ at a time k. Note that we can also transform this task into one of mapping if we have an accurate estimate of the trajectory for time $0tok$, that is compute the probability $P(\boldsymbol{m}|\boldsymbol{X}_{0:k}, \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k})$. In the same way we could formulate it into a localization problem, assuming that the landmark locations are roughly known and compute the conditional probability of the robot trajectory $P(\boldsymbol{X}_{0:k}|\boldsymbol{m}, \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k})$.

Appendix B

Software setup

In the current section the basic tools and software that need to be installed on the robotic agents in order for the SLAM execution to take place successfully are outlined.

Warning: Notice that the packages mentioned exist both on `github.com` as well as the server of the Control Systems Lab. For the stable version of the packages users should refer to `http://controlsystemslab.gr/code/bergercookie`. However, for further development, issue tracking as well as more frequent updates users should use packages of `http://github.com/bergercookie`.

B.1 MRPT Installation

Detailed instructions on installing MRPT are given here[1]. However, to get the latest version of the graphSLAM algorithm developed it is strongly suggested that user downloads MRPT from the corresponding github fork[2] and installs it from source. The latter can be automated using this script, that sets up MRPT from the designated github repo along with all its binary dependencies. [3]

B.2 Installation of ROS Packages

For information, installation instructions on the individual packages refer to the catkin_ws[5] repository. The latter includes the sources of all the required packages needed to run graphSLAM successfully both in simulations as well are real-time setups [6]

Instructions on actually running single- or multi-robot graphSLAM either in simulations or in real-time experiments are provided in their corresponding sections (5.3.1, 3.5.1).

[1]`http://www.mrpt.org/MRPT_in_GNU/Linux_repositories`

[2]`http://github.com/bergercookie/MRPT`

[3]Installing from source, requires that the wxWidgets 3.0 version is used instead of 2.8 (suggested by the corresponding installation instructions here[4].

[5]`https://github.com/bergercookie/catkin_ws`

[6]In case overall compilation fails, or a CMake dependency is missing, open an issue describing the problem in the Github page[7].