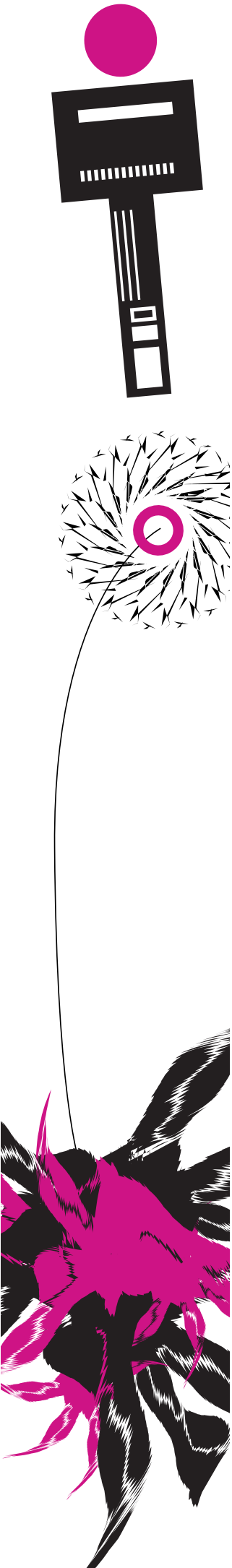

RAM ROBOTICS AND MECHATRONICS

# COMPONENT-BASED SLAM IN RTAB-MAP

A. (Ángel) Lorente Rogel

MSC ASSIGNMENT

**Committee:**
dr. ir. J.F. Broenink
dr. ir. D. Dresscher
dr. ir. G.A. Folkertsma, CEng
dr. V.V. Lehtola

October, 2021

067RaM2021
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

# Acknowledgements

Although it is just my name on the cover, many people have contributed to this research both technically and beyond academical matters, and for that I owe them my gratitude. I would like to dedicate this page to all the people without whose support this project would have never happened.

Firstly, to my supervisors dr. ir. Douwe Dresscher and dr. ir. Geert Folkertsma who proposed me this project and guided me to get the best out of it regardless the difficulties of the pandemic. They allowed me to pursue my dream of a career in mobile robots, and I am very grateful for this opportunity.

To my family, Leonor, Ángel and Sandra, who supported me beyond my master and always believed in me. I am extremely thankful for having a loving mother, an affectionate father and an uplifting sister who are always there for me when I need them.

To my friends who always stayed by my side and gave me many moments of joy, no matter the distance. Last but not least, to Carmen, who was always there with me, during the many ups and lows in this journey, and gave me the strength and support when I needed them the most.

# Summary

SLAM is widely used in the development of new autonomous robots. It has many applications like: self-driving cars, autonomous cleaning robots, inspection, exploration, etc. Nowadays, there are several open-source SLAM frameworks that contribute to the research of new technologies. However, the reusability and adaptability of most of the frameworks is very limited. As a consequence, the implementation of new sensors and algorithms requires a significant effort, adding even months to every new development.

This project studies modular SLAM, which aims to improve the reusability of SLAM. Most of the related work focus on creating completely new frameworks, which reduces the chances of other researchers and developers to use their designs. For this reason, the main goal of this project is to design and implement a modular SLAM framework, compatible with the state of the art open-source solutions. For this task, RTAB-Map was selected, which is widely used by the community and previously studied in the RaM department.

In this project, the modularity issues in RTAB-Map are analyzed to find a possible solution. The main contribution of this project is the design and implementation of a modular SLAM framework, capable of working with RTAB-Map. The design consists of a back-end, which performs the tasks of creating, storing and optimizing the pose-graph. In order to create the pose-graph, the back-end is capable of communicating with an arbitrary number of front-ends. Every front-end is provided with a set of commands to create the entities of the graph. The framework is sensor agnostic and can easily work with any kind of sensor system that uses pose-to-pose or pose-to-landmark constraints.

During this project it was also implemented a client-server UDP communication protocol to allow inter-process communication, a ROS interface and three front-end modules for processing odometry and Apriltags landmark data. These modules, together with a visualization tool are used for testing the proposed design in different situations with different sensor data. The proposed framework proves to be able of using the information generated by RTAB-Map, and improve the trajectory estimation when adding more sensor systems.

The usage of the modular SLAM framework together with RTAB-Map allows a much faster implementation of new sensors and algorithms while being capable of using most of the functionality in sensor processing available in RTAB-Map.

# Contents

# Abbreviations

**BoVW** Bag of Visual Words.

**BoW** Bag of Words.

**Dof** Degrees of Freedom.

**EKF** Extended Kalman Filter.

**F2F** Feature to Feature.

**F2M** Feature to Map.

**GPS** Global Positioning System.

**ICP** Iterative Closest Point.

**ID** Identification.

**IMU** Inertial Measurement Unit.

**LiDAR** Laser imaging, Detection, And Ranging.

**LO** LiDAR Odometry.

**LTM** Long Term Memory.

**MAP** Maximum A Posteriori.

**NDT** Normal Distribution Transform.

**PF** Particle Filter.

**PnP** Perspective from n Points.

**RaM** Robotics and Mechatronics.

**RANSAC** Random Sample Consensus.

**ROS** Robot Operative System.

**RTAB-Map** Real-Time Appearance-Based Mapping.

**S2M** Scan to Map.

**S2S** Scan to Scan.

**SFM** Structure From Motion.

**SLAM** Simultaneous Localization And Mapping.

**STM** Short Term Memory.

**VO** Visual Odometry.

**WM** Working Memory.

# 1 Introduction

## 1.1 Context

The SLAM (Simultaneous Localization and Mapping) problem has been object of study for several decades, and is nowadays considered as a mature research field with many applications in robotics. SLAM is known as the process of creating a map of an environment while estimating the position of the robot in the same map. The main challenge of this problem resides in the simultaneous estimation of both information, but nowadays there are tons of studies that propose theoretical solutions for SLAM.

The ability of a robot for solving the SLAM problem proved to be very useful in many fields such as exploration, warehouse inspection (like the Spot robot from Boston dynamics (2021)), autonomous cleaning (like the SB2 from Aziobot (2021)), self-driving cars (like the model Y from Tesla (2021)) and many others. The industry is implementing more and more robots with semi-autonomous capabilities and there is a growing tendency towards fully-autonomous implementations. For this reason, there is a great interest in improving the existing algorithms and creating new ones to achieve better performance in SLAM.

## 1.2 Problem statement

Commercial robots, like the ones mentioned in the previous section, may use very different sensor setups depending on the problem at hand. Furthermore, in the robotics community there is a constant search for new data sources and sensor systems that could increase measurement's accuracy or reduce production costs. This great variety of sensor platforms creates the need of adapting the SLAM implementations to support the different kinds of input data.

There are many open-source SLAM frameworks available, which offer solutions for a certain set of sensors and algorithms. However, most of these frameworks were not designed to be reused. More often than not, open-source SLAM solutions are made available to be tested by other researchers or industry developers, and not for being modified. For instance, the documentation given in popular SLAM frameworks like RTAB-Map (Labbé and Michaud (2019)) and Google Cartographer (Hess et al. (2016)) focus on how to use the framework as a whole, showcasing the available parameters.

Although some frameworks like RTAB-Map still receive improvements adding support for different kind of sensor arrangements, and SLAM solutions, the architecture of these systems is not suited for other researchers to implement their own algorithms. The main reason for this is the size and complexity of the code, together with the lack of official documentation that makes it time consuming to even know where to start the modifications. Additionally, as shown in the work of Mark te Brake (2021), open-source SLAM frameworks can contain numerous dependencies between modules that make more complex the modification of individual modules.

This translates into *"monolithic"* designs that require a lot of time to implement new features or to add support to new sensors. This problem delays the creation of new algorithms, because there is no framework on which they could easily implement and test new approaches.

The ideal SLAM system would present a structure formed by well-defined "blocks" and "interfaces" allowing the exchange of a part whenever is needed while maintaining the rest intact. Such an structure is difficult to obtain due to the great number of robots with different hardware and goals that exchange and store different kinds of data. Each SLAM algorithm was designed having in mind different specifications (like the sensory system or the computation resources) and this is where the complexity of creating a modular SLAM system capable of encompass the current implementations and the ones that might come resides.

### 1.3 Related work

Several authors noticed the importance of a modular SLAM framework and worked towards this goal. Work relevant to this project can be categorized into: analyses that focus on the general structure of the estimation problem, SLAM algorithms on the different paradigms and proposals of modular frameworks.

Initially the classical SLAM solutions were filter-based, and some authors focused on implementing different types of sensors in filter-based approaches. Lynen et al. (2013) is an example of where a framework is created based on an iterated EKF focusing on the sensor fusion of a heterogeneous sensor system. This sensor combination is achieved using an expansion of the state vector adding new states and biases, depending on the number and kind of sensors that are used. Another attempt using a filter-based approach, this time using particle filters is given in Tim Broenink (2016). In this study, the author creates an structure based on interfaces that connect five functional blocks present in every SLAM system: robot, environment, feature, sensor and world specific.

Jeroen Minnema (2020) is a more recent approach which tried to find similarities between the three most popular SLAM: EKF SLAM, PF SLAM and graph-based SLAM. Minnema's work is a step forward on the work of Mohamed A. Abdelhady (2017), where a thorough analysis of the SLAM problem is given, and identifies three necessary elements for solving the SLAM problem: the sensor's measurement, a model (measurement model or motion model with its Jacobian) and a noise model. In Minnema's work, also a modular framework for the SLAM problem is proposed. The framework consists of a "back-end" that uses these three inputs depending on the kind of sensor used to solve the SLAM problem. The different sensors are categorized in idiothetic, absolute allothetic and relative allothetic. The difference between idiothetic and allothetic is that the idiothetic sensors use intrinsic data of the platform like encoders, which incrementally estimate the position of the robot, whereas the relative allothetic sensors use external features or landmarks to estimate the position. The difference between absolute and relative allothetic is the coordinate frame used, i.e an absolute allothetic sensor could be a GPS system. These sensor definitions allows the same treatment for the same kind of sensors improving the reusability, but this implies that every sensor and technique must be categorized into these definitions in order to be used.

In Joost van Smoorenburg (2020), it is studied how to implement Minnema's design into RTAB-Map, analyzing which aspects of the RTAB-Map algorithm should be changed in order to work with Minnema's framework. Smoorenburg's work gives a good first insight on how the implementation of popular SLAM algorithms are not suited for modularity and how the components of RTAB-Map could be mapped into a modular framework.

Mark te Brake (2021) also looks into the functional structure of popular state of the art algorithms. In his work, the data flow of two visual graph-based SLAM algorithms, RTAB-Map and ORB-SLAM2, is analyzed. This study tries to find a common structure between both implementations that could be generalized to other solutions as well. As a result of this analysis, Mark's work also proposes a set of components that could be used on a modular framework. Nonetheless, the proposed design is limited to a visual SLAM algorithm and it is not stated how these components would interface with different sensor types like range sensors.

On the last two decades, the graph-based SLAM approach became very popular and several authors implemented solutions using this paradigm, improving its performance. Grisetti et al. (2010) provides with an overview of the graph-based SLAM system, giving a coarse distinction between the algorithms in charge of creating the graph (named "front-end") and the methods that focus on the optimization of the graph (or "back-end"). Both blocks are connected and exchange the necessary information. Note that the back-end needs the pose-graph in order to optimize it, and the front-end benefits from the optimized poses to get better estimations.

Using the front-end back-end architecture explained in Grisetti's work, authors like Colosi worked on the reusability for each block. Colosi focused on a multi-sensor framework that could maximize the reusability of the front-ends. In Colosi et al. (2019) and Colosi et al. (2020), recurrent patterns in the context of graph-based SLAM are analyzed, designing a front-end model composed by modules and sub-modules. The core *modules* serve as abstract layers representing the main tasks in a SLAM algorithm, like loop closure detection and "*data-aligning*", and they are composed by *sub-modules* which perform the specific tasks, like data association, for the different data types. This way, independently of the implementation or exact approach provided by the combination of *sub-modules*, the input-output data flow of a core module remains the same. On the other hand, the *sub-modules* are the ones that control the specific behaviour of the core modules, and their implementation need to be modified to add new features or support new sensors.

There are also studies like Blanco-Claraco (2019) which focus on creating a framework composed by a set of predefined modules: back-end, map storage, sensor data acquisition, front-ends and visualizers. In this work, they do a quick analysis on the state of the art of front-end solutions and models. Based on this they implement a framework defined as a set of virtual classes so the users can define their own modules. This work also remarks the importance of a pose-graph that represents a well determined system, so it can be properly optimized. The author proposes a solution integrated in the back-end to prevent an indeterminate system using a kinematic model so all the nodes in the graph have at least one edge. Blanco also implements some front-end modules for the usage of LiDAR, visual and IMU data .

## 1.4 Goals

All the related work contribute to the generation of a modular SLAM framework. However, most of the authors do not directly tackle the implementation of their design into popular open-source solutions. This fact significantly reduces the chances of other researchers and developers to use their designs. For this reason, a major goal in this project is to have a solution compatible with the open source SLAM implementations. For this task, RTAB-Map was selected, which is widely used by the community and previously studied in the RaM department.

The aim of this project is to design a modular SLAM framework, clearly defining the interfaces and the data needed to solve the SLAM problem. The main goal can then be defined as:

- Design and implement a modular SLAM framework that is compatible with state of the art solutions and eases the implementation of new sensors and algorithms.

Therefore, the main goal can be divided into several research questions which also serve as milestones to maintain a well defined structure during the project:

1. What are important aspects of modularity? How is modularity applied to SLAM?
2. What aspects of the current implementation of RTAB-Map are not compatible with modularity? How could these be modified to increase the reusability of the framework?
3. Which modular framework in the state of the art would best fit the structure of RTAB-Map?
4. What interface would easily allow the implementation of new sensors and algorithms into the framework?

## 1.5 Outline

The rest of the report will be structured as follows. In chapter 2, all the concepts needed to understand this project are briefly presented, also pointing to relevant literature for the interested reader. Firstly, in Section 2.1, the general concepts of Software reusability are reviewed. In 2.2, the core concepts and motivation for graph-based SLAM are given. Finally in Section 2.3, the popular SLAM framework RTAB-Map is presented, going through its architecture.

Chapter 3, starts with the analysis of the related work to find a suitable architecture for RTAB-Map. In Section 3.2, the modularity issues in RTAB-Map are thoroughly analyzed, which serves as a prelude for Section 3.3 where a brief motivation for the design is given. This motivation justifies the design decisions made for the framework proposal presented in Section 3.4. The framework is implemented into RTAB-Map as explained in Section 4.2 and with other algorithms using the ROS interface as shown in Section 4.3.

Chapter 5, explains the environment in which the framework was tested, the simulations, metrics and the setup for the different experiments. In Chapter 6, it is given some final thoughts for this project and some recommendations for the further development of the framework.

# 2 Background

In this chapter, a brief review of the concepts that are basic to understand the proposal of this project is given. These topics are broadly explained in the literature and it is highly recommended to read the referenced bibliography for a more in-depth description of each topic.

## 2.1 Software reusability

The concept of reusability in computer science and software engineering is the use of existing assets within the software development process. Prior to analyzing modularity specifically for SLAM systems, it is necessary to set the foundations of reusability in general, for every software implementation. This is a very recurrent topic and there are several references in literature (i.e Dennis Ellery (2017)). For this reason, in this section it is summarized the most important concepts about software reusability.

- **Abstraction**: Abstraction refers to selecting which details are shown and which are hidden in the different levels. The main purpose of abstraction is to only focus in what is relevant for each level. This is also a difficult task, as the designer must decide what is important to provide with the necessary functionality. Abstraction highly reduces the complexity of the implementations, contributing to their reusability.

- **Composition and variability mechanisms**: Composition is the process of interconnecting different modules to construct a system. It relies in the definition of well-defined interfaces and self-contained modules. Composition allows the configuration of the system into different modes and can be used in combination with variability mechanisms to provide with more flexibility. Variability mechanisms refers to the possibility of changing certain properties of a component, to customise it or modify it for reusing it.

- **Hierarchy**: This practice refers to the division of a big problem or task into smaller elements. This division is usually performed in different layers or hierarchies, in which the different modules are allocated depending on their role or specific task. The use of a hierarchy contributes to the generation of specialized elements that are easier to reuse.

- **Interfaces**: An interface defines the limits of a module, what functionality offers and what kind of information needs as input to operate properly. The most important role of the interfaces is to control the data flow between modules, restricting any unnecessary dependencies and keeping them at the bare minimum. Interfaces also manage the access to data, so other modules only access the allowed data, using what is commonly known as *getters* and *setters*.

## 2.2 Graph-based SLAM

Graph-based SLAM is one of the paradigms for solving the SLAM problem together with Kalman filters and Particle filters. This concept was firstly brought by Lu and Milios (1997), who proposed a global optimization of the error generated by the constraints using a least-squares approach. Followed by Gutmann and Konolige (1999), who found an effective way of generating the constraint-based network adding loop-closures in a LiDAR-based system. This paradigm has gained popularity in the last two decades because of the advancements in sparse linear algebra that allowed more efficient solutions to the optimization problem.

The idea is to use a graph to represent the SLAM problem, highlighting its spatial structure. This graph is also known as a pose-graph, and is formed by two main components:

- **Nodes**: Represent robot poses (i.e (x, y, $\theta$) in 2D or (x, y, z, pitch, yaw, roll) in 3D; please note that other coordinate systems could be used). Nodes are usually generated using an heuristic (i.e a displacement of more than 5 meters from the previous node) while the robot is operating. *Landmarks*: Landmarks are identifiable features in the environment. They are similar to nodes in that they store spatial information (i.e x, y), but they do not store orientation information, so the edges that connect landmarks and robot poses are different.

- **Edges**: Between two nodes exists a spatial constraint (also called virtual constraint) which is extracted from the sensor data. Constraints are represented as edges in the pose-graph and labeled with the uncertainty introduced by the sensor system. Edges are created in two cases

  - *Odometry-based*: when new odometry data is available (i.e using wheels' encoders, incremental scan matching, etc).
  - *Observation-based*: when the robot revisits a place that was already stored in the graph, also known as a loop-closure (i.e using visual feature matching or ICP).

  Edges can be represented as transformations using homogeneous coordinates '$z$' and an information matrix '$\Omega$' (inverse of the uncertainty matrix), which models how much can we trust that constraint.

Graph-based SLAM decouples the SLAM problem into two main tasks: the graph construction (which is usually performed by the front-end) and the graph optimization (or back-end).

### 2.2.1 Front-end

The front-end is the one in charge of extracting the relevant information from the sensor data and its design is heavily sensor-dependent. The front-end is in charge of tasks like: feature extraction and feature matching (if applicable), data association and local optimization. As seen in section 1.3, there are studies like Colosi et al. (2019) and Colosi et al. (2020), in which they study the creation of a modular front-end to maximize the reusability of the code. Please refer to Section A for a more detailed review in the state of the art in front-ends.

### 2.2.2 Back-end

Contrary to the front-end, the back-end is sensor agnostic and relies on an abstract representation of the data. Graph-based SLAM formulates the problem as a Maximum A Posteriori (MAP) estimation problem (Cadena et al. (2016)) and the back-end is responsible for this estimation.

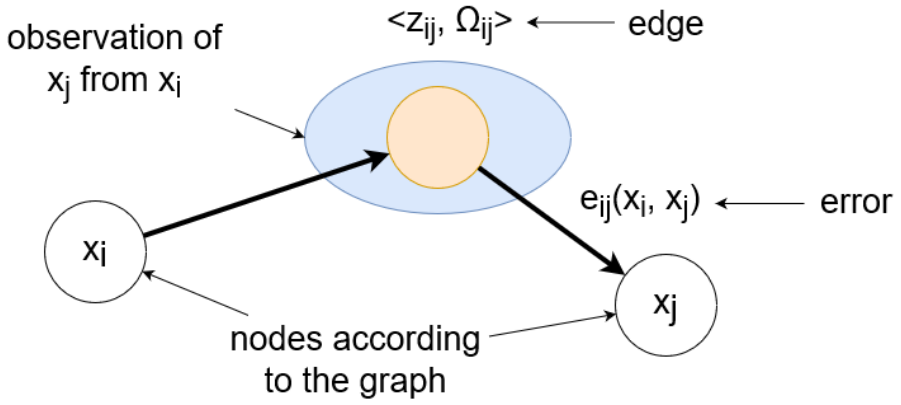


**Figure 2.1:** A simple pose-graph optimization example.

To understand the optimization process let's review the example in Figure 2.1. Where it is shown a simple pose-graph example defined by two nodes $x_i$ and $x_j$ connected by a constraint generated by a common observation between the two nodes. The error function between both nodes is represented by the difference between what the current configuration of the pose-graph states of $x_j$ and what the observation states. The error can be mathematically defined as (2.1), where $(X_i^{-1} X_j)$ represents the expected observation from $x_i$ to $x_j$ given the current configuration of the graph and $Z_{ij}$ the mean of the virtual constraint. Note that is in capital letters to represent matrix form, and t2v a function that transforms to vector form.

$$\mathrm{e}_{ij}(\mathrm{x}) = \mathrm{t2}v\left(\mathrm{Z}_{ij}^{-1}\left(\mathrm{X}_i^{-1}\mathrm{X}_j\right)\right) \tag{2.1}$$

The goal of the optimization process is to minimize this error for the whole state vector, which is found as the minimum to the negative log likelihood of all observations (2.2). Where $x$ is the state vector formed by the concatenation of all the nodes (2.3). This conclusion is well-known by the community. A more thorough analysis of the probabilistic formulation of the approach is given in Grisetti et al. (2010).

$$\mathrm{x}^* = \underset{\mathrm{x}}{\operatorname{argmin}} \sum_k^n \mathrm{e}_k^T(\mathrm{x})\Omega_k \mathrm{e}_k(\mathrm{x}) \tag{2.2}$$

$$\mathrm{x}^T = \begin{pmatrix} \mathrm{x}_1^T & \mathrm{x}_2^T & \cdots & \mathrm{x}_n^T \end{pmatrix} \tag{2.3}$$

The solution to (2.2) can be found using algorithms like Gauss-Newton or Levenberg-Marquardt. Modern solvers exploit the sparse nature of the pose-graph to solve the optimization problem efficiently. Open access libraries like GTSAM (Dellaert (2012)), $G^2o$ (Kümmerle et al. (2011)) and Ceres (Agarwal and Mierle (2010)) are capable of solving this kind of problems in a few seconds.

## 2.3 RTAB-Map: An open-source SLAM solution

RTAB-Map is a popular graph-based SLAM approach which implements a visual loop closure detection using an incremental BoW. This algorithm can take LiDAR, RGB-D and stereo camera data as input, allowing different operation modes like 6 Dof and 3 Dof mapping. One of RTAB-Map's main contributions is the implementation of a memory management system that allows a real time operation on large-scale environments while maintaining long-term mapping. RTAB-Map is also compatible with ROS and had a great support from the scientific community in the last decade.

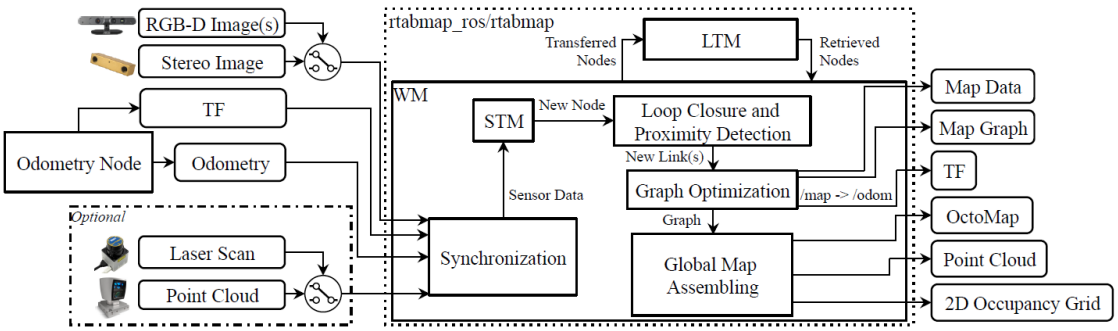


**Figure 2.2:** RTAB-Map ROS node block diagram Labbé and Michaud (2019).

In Figure 2.2 the different functional blocks of RTAB-Map can be appreciated, as presented in the author's work (Labbé and Michaud (2019)). Here it is clear that the algorithm needs at least an odometry source and either an RGB-D or Stereo image input. The Laser sensor input serves as an optional range data input which can be either a 2D laser scan or a 3D pointcloud and can be used to form 2D or 3D occupancy grids respectively. RTAB-Map also includes an odometry node in case no external odometry is provided. The main block of RTAB-Map contains the memory management system and the blocks of loop closure, proximity detection, the graph optimization and the global map assembling modules. RTAB-Map is capable of giving several outputs in real time: the generated map in different formats (OctoMap, occupancy grid and 3D pointcloud), the odometry correction in tf form (ROS format for transform) and the generated graph with and without the sensor information.

### 2.3.1 Odometry node

RTAB-Map implements an odometry node that is able of computing visual odometry (VO) or LiDAR odometry (LO) in case that there is no external source of odometry i.e wheels odometry or other source of VO or LO. This node computes odometry based in the methodology presented in Scaramuzza and Fraundorfer (2011), computing F2F and F2M in case of VO and S2S and S2M in case of LO.

#### Visual odometry

For the visual odometry approach, the transform between the camera coordinate frame and robot frame and an stereo or RGB-D image source are needed. The input image is used to obtain visual features. These will later go through a feature matching step comparing the features obtained in the current image frame with the previous key frames or the stored feature map, F2F and F2M respectively. Once the features are matched, a motion prediction is computed using PnP RANSAC, which is later optimized with a local bundle adjustment. Finally, the optimized pose with its uncertainty is given as an output of the odometry node. In case the new pose is sufficiently significant (determined by a threshold on the number of inliers in the motion estimation), the extracted features are added to the feature map for later feature matching. Note that the feature map has a maximum number of stored features after which the older features are discarded, so this is not a long-term mapping algorithm.

#### LiDAR odometry

The LiDAR odometry is very similar to the one presented in the visual odometry. In this case laser sensor data should be given as input together with the transform between the LiDAR coordinate frame and the robot's base frame. In addition external odometry like wheels odometry could be added to improve motion prediction as LiDAR odometry can lose its track if not enough features are detected. The first step is to filter the pointcloud data, after it proceeds with ICP registration of the current pointcloud in the previous key frame or in the map, S2S or S2M respectively. In this case the map is a pointcloud formed by all the registered scans of each key frame. After the ICP registration, the next odometry pose is obtained and given as an output in the form of an updated transform and a pose with uncertainty. Similarly to the VO approach, if the correspondence ratio of the current frame is under a predefined threshold, the new frame is considered as a key frame.

### 2.3.2 Graph creation and data storage

As mentioned above, RTAB-Map is a graph-based SLAM system. This means that the system recreates the environment using nodes and links.

**Nodes and links in RTAB-Map**

Nodes are certain instants in time in which the sensor data is received. The frequency of the sensors mounted in the robot determine the frequency of creating new nodes. In RTAB-Map, all the input data is synchronized and stored in the same node, which will be now called signatures, according to Labbé and Michaud (2018) each signature may store (depending on the available sensor data):

- ID: Unique time stamp.
- Weight: Importance of the signature. Used for the memory management system.
- BoW: Visual words used for loop closure detection and weight update. This is also known as the image's signature.
- Occupancy grid (Labbé and Michaud (2019)).
- Sensor data:
  - Pose: Odometry input.
  - RGB image: The one used to obtain the features.
  - Depth image: Used to find 3D position of the visual words.
  - Laser scan: Used for loop closure transformations and odometry refinements, and by the Proximity Detection module.

All the signatures are interconnected by the so called "links", which represent the rigid body transform between the signatures and the uncertainty of the measurement as an information matrix. These links can be stored in 3 Dof (x, y, $\theta$) or 6 Dof (x, y, z, pitch, yaw, roll), depending if the space is represented in 2D or 3D, note that Euclidean coordinates are used. All the links contain the same information, but RTAB-Map has a naming convention depending on the module that provided these links. The types are:

- Neighbour link: Created between a new signature and the previous one.
- Proximity link: Added when two close signatures are aligned together using LiDAR data and scan alignment.
- Loop closure link: Added when a loop closure is detected between the new signature and one in the map.
- Landmark constraint: This link is not explicitly defined in RTAB-Map's official documentation, but it is present in the new features of the algorithm. This may not be a link in the formal definition that we have, but the constraint that connects a signature with an identifiable landmark. This is possible with Aruco markers or April tags, which detection is possible in the last versions of RTAB-Map since 2019.

Note that the GPS data is not stored as a link. The reason for this is that the GPS data is only used as a prior estimation of a node. All the links are used as constraints for graph optimization. Every time a loop closure or proximity link is created, the graph is optimized to reduce the odometry drift. The information of the pose-graph is extracted from the net of signatures and links and given to the optimizer.

**Memory management**

The memory management system runs on top of the RTAB-Map's core. This system is in charge of managing how the pose and sensor data is stored and selecting which one is used to perform proximity detection and loop closure detection. The memory management system is divided in three main memories: Short-Term Memory (STM), Working Memory (WM) and Long-Term Memory (LTM). As pointed out in Mark te Brake (2021), the two main goals of this distinction is: to separate the recently obtained data from the one used in loop closure and to keep a low number of signatures during loop closure detection to remain in real time constraints.

- STM: It can be seen as a fixed size buffer that processes and stores the most recent signatures, so they do not affect loop closure detection. According to Labbé and Michaud (2019), here is where the occupancy grid is computed and all the information of the signatures is assembled. According to Labbe and Michaud (2013), there is a previous memory called Sensory Memory (SM) in which the features extraction and feature reduction is performed before entering STM. The extracted features are quantized into visual words using an incremental BoW (BoW).

- WM: This memory contains all the signatures that are candidates for a loop closure. This memory is usually stored in the so called RAM memory, so only the signatures that are not candidates for *transfer* remain in the WM to reduce memory usage and processing time.

- LTM: This memory stores all the rest of signatures that are neither recent nor candidates for loop closure. According to Labbe and Michaud (2013) the signatures are stored in a database containing the link, the signatures' ID and their signature.

The memories are managed by three main methods that decide whether a signature is stored in one memory or another:

- Rehearsal: This method operates on top of the STM, reducing the amount of signatures that enter the WM and updating the weights. This method uses the signatures to determine whether two signatures are too similar. If they are, then it fuses the data and updates the weight of the signature.

- Retrieval: This is one of the methods that operates between the WM and the LTM. Once an hypothesis of loop closure is accepted, the neighbouring signatures with higher loop closure probability are retrieved from LTM to WM. This method allows the system to update the WM with signatures that are candidates of a loop closure.

- Transfer: Opposite to the "Retrieval" method, this one sends the less significant signatures to LTM for long-term storage. This method contains the criteria to evaluate which signatures to transfer, that is based in two heuristics (Labbé and Michaud (2018)): the older signatures with less weights have priority to be transferred to LTM and the signatures that are used in path planning must remain in WM.

### 2.3.3 Loop closure detection

The loop closure detection in RTAB-Map is based in the commonly known approach of the BoVW. In this methodology, the features of every detected key-frame are combined into the BoW, and every key-frame is quantized into a representation using the resulting visual words. A discrete Bayesian filter is used to keep track of the loop closures by estimating the probability that the current location has with the signatures available in WM. If the probability is higher than a certain threshold, this hypothesis is considered as a loop closure candidate and the pose-graph is updated in consequence.

### 2.3.4 Optimization

In RTAB-Map, the world model inside WM is optimized to accurately represent the trajectory performed by the robot. To perform the optimization, the pose information is extracted from the signatures and introduced into a nonlinear optimizer. Currently RTAB-Map implements several of the most popular optimizers like g2o, gtsam and TORO. The optimization is carried out as a pose optimization, which is described in Section 2.2.2.

### 2.3.5 Dependencies in RTAB-Map's architecture

RTAB-Map's implementation has a lot of dependencies between modules, which severely affects its reusability. In Mark te Brake (2021) they did a thorough analysis about these dependencies and it was of great help to understand RTAB-Map's architecture for this project. In this Section, I reproduce some of his figures and paraphrase the dependencies he found, as they will be used later in the Analysis.

In Figure 2.3, it is shown the dependencies of the Signature creation module described in Table 2.1. It is possible to see that this module needs of appearance data and odometry input. The Signature creation module also interfaces with the pose-graph optimization and Transfer modules to notify about the changes in memory and the STM to story the newly created Signatures.

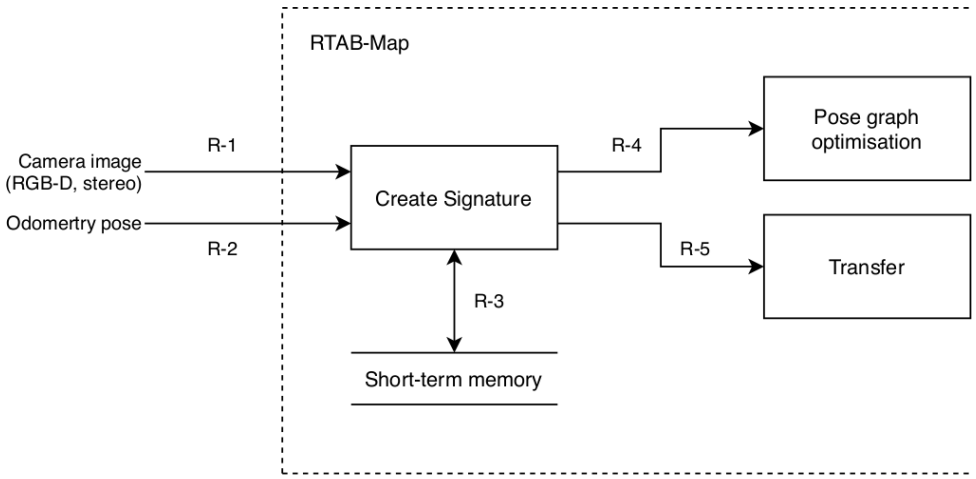


**Figure 2.3:** Dependencies present in the signature creation module (image from Mark te Brake (2021)).

| Dependency | Direction | Description |
|---|---|---|
| R-1 | In | Visual data. |
| R-2 | In | Odometry pose. |
| R-3 | In | Previous signature's data for linkage with the new one. |
| R-3 | Out | Storing the new signature. |
| R-4 | Out | Flag: Storage data was modified. |
| R-5 | Out | Number of the newly added signature. |

**Table 2.1:** Description of the dependencies present in the signature creation module.

The Rehearsal module sends similar kind of information to the pose-graph optimization and Transfer modules notifying about the modification of Signatures in storage. This modules also exchanges information with the STM to gather information of the two most recent Signatures or delete the one that considers that has no sufficient novel information. These dependencies are shown in Figure 2.4 and described in Table 2.2. In this figure it is also shown the dependency between the STM and WM moving the oldest Signature after the Rehearsal.

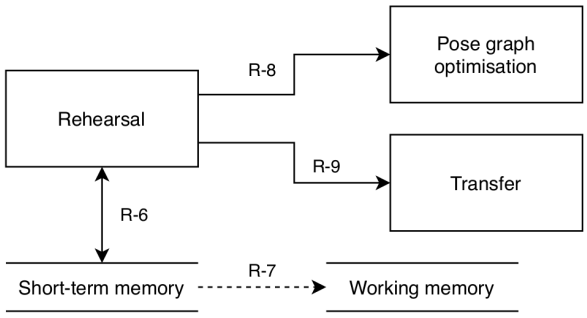


**Figure 2.4:** Dependencies present in the Rehearsal module (image from Mark te Brake (2021)).

| Dependency | Direction | Description |
|---|---|---|
| R-6 | In | Data of the newest and the previous Signature. |
| R-6 | Out | Delete the newest Signature and update its weight if necessary. |
| R-7 | - | Move the oldest Signature to the Working memory after the Rehearsal. |
| R-8 | Out | Flag: Storage data was modified. |
| R-9 | Out | Number of the deleted Signature. |

**Table 2.2:** Description of the dependencies present in the Rehearsal module.

In Figure 2.5 are shown the dependencies in the Bayes filter module, described in Table 2.3. This module gets Signature data from the STM and the WM looking for a loop closure, and notify the Retrieval and Loop closure link generation modules when a loop closure is detected.

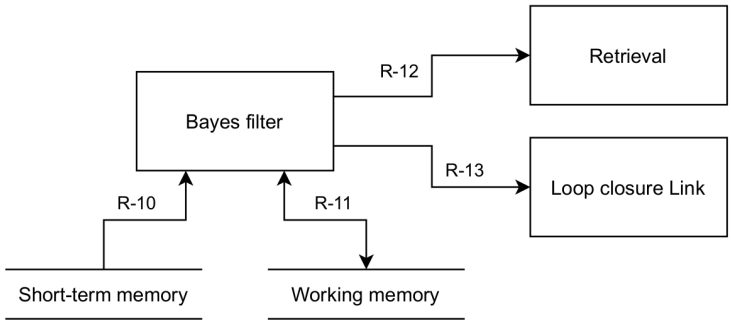


**Figure 2.5:** Dependencies present in the Bayes filter module (image from Mark te Brake (2021)).

| Dependency | Direction | Description |
|---|---|---|
| R-10 | In | Appearance data of the newest Signature. |
| R-11 | In | Appearance data of all the Signatures in WM. |
| R-11 | Out | Virtual Signature Storage. |
| R-12 | Out | This Signature is a candidate for a loop closure. |
| R-13 | Out | This Signature forms a loop closure with the current Signature. |

**Table 2.3:** Description of the dependencies present in the Bayes filter module.

The dependencies of the Retrieval module are shown in the Figure 2.6 and described in the Table 2.4. The Retrieval module receives information from the Bayes filter module, the STM and the WM to select which Signatures are retrieved from the LTM. Once a Signature is retrieved, it is deleted from the LTM and stored into the WM. Afterwards, the pose-graph optimization and Transfer modules are notified about the Retrieval, triggering a possible graph optimization and forbidding the transfer of the Signatures that were recently retrieved.

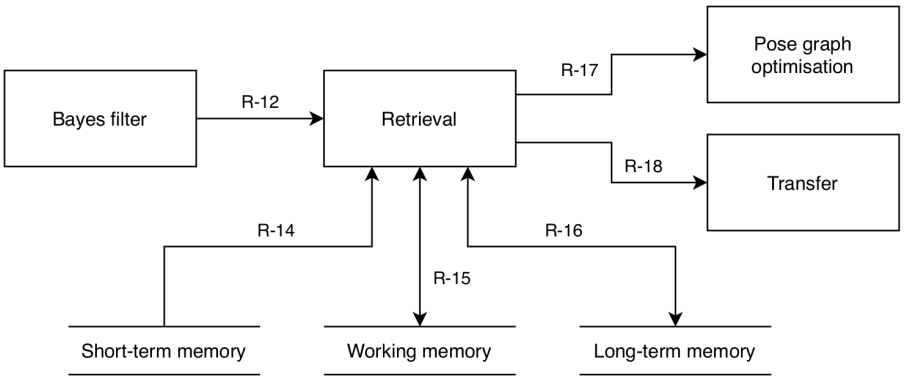


**Figure 2.6:** Dependencies present in the Retrieval module (image from Mark te Brake (2021)).

| Dependency | Direction | Description |
|---|---|---|
| R-12 | In | This Signature is a candidate for a loop closure. |
| R-14 | In | Data of Signatures in STM for selecting a possible retrieval. |
| R-15 | In | Data of Signatures in WM for selecting a possible retrieval. |
| R-15 | Out | Store the retrieved Signatures from LTM. |
| R-16 | In | Data of Signatures in LTM for selecting a possible retrieval. |
| R-16 | Out | Delete Signatures retrieved from LTM. |
| R-17 | Out | Flag: Storage data was modified. |
| R-18 | Out | Amount of retrieved Signatures. List of Signatures that can not be transferred to LTM. |

**Table 2.4:** Description of the dependencies present in the Retrieval module.

The loop closure link generation module receives the candidate ID from the Bayes filter module, retrieves the pose data of corresponding Signature from the WM and the current Signature from the STM. Once the loop closure link is computed, the involved Signatures are updated with the new link and a flag is sent to the pose-graph optimization module to trigger the optimization. These dependencies can be seen in Figure 2.7 and described in Table 2.5.

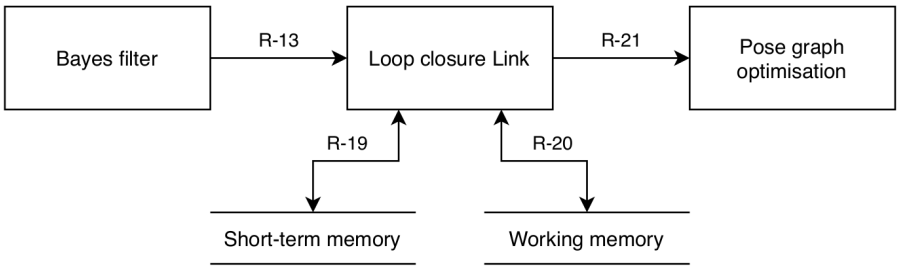


**Figure 2.7:** Dependencies present in the loop closure module (image from Mark te Brake (2021)).

| Dependency | Direction | Description |
|---|---|---|
| R-13 | In | This Signature forms a loop closure with the current Signature. |
| R-19 | In | Current Signature data for computing the loop closure constraint. |
| R-19 | Out | Store the loop closure Link. |
| R-20 | In | Signature candidate for loop closure for computing the loop closure constraint. |
| R-20 | Out | Store the loop closure Link. |
| R-21 | Out | Flag: Storage data was modified. |

**Table 2.5:** Description of the dependencies present in the loop closure module.

The dependencies of the pose-graph optimization module are shown in Figure 2.8 and described in Table 2.6. The pose-graph optimization module receives flags from the Signature generation, Rehearsal, Retrieval and loop closure link generation modules to trigger the optimization, and the pose-graph information from the STM and the WM. Once the graph is optimized, pose-graph optimization module sends the information of the optimized poses to the STM and the WM to update the Signatures.

| Dependency | Direction | Description |
|---|---|---|
| R-4 | In | Flag: Storage data was modified. |
| R-8 | In | Flag: Storage data was modified. |
| R-17 | In | Flag: Storage data was modified. |
| R-21 | In | Flag: Storage data was modified. |
| R-22 | In | Data of Signatures in STM to construct a pose-graph. |
| R-22 | Out | Store optimized poses. |
| R-23 | In | Data of Signatures in WM to construct a pose-graph. |
| R-23 | Out | Store optimized poses. |

**Table 2.6:** Description of the dependencies present in the pose-graph optimization module.

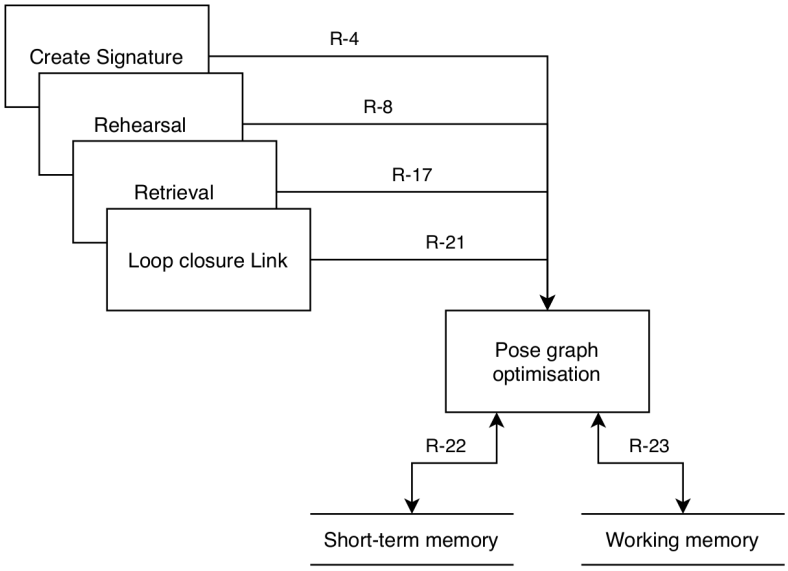


**Figure 2.8:** Dependencies present in the pose-graph optimization module (image from Mark te Brake (2021)).

Finally, in Figure 2.9 are shown the dependencies in the Transfer module, described in Table 2.7. The Transfer module receives information from the Signature generation, Rehearsal and Retrieval modules to decide which Signatures can't be transferred. The information of the elapsed time of an iteration is used to decide if a Signature needs to be transferred. If the elapsed time is larger than a certain threshold, the module receives data of the Signatures in the STM and the WM to select which Signature needs to be transferred and deleted the selected Signature from the WM to store it in the LTM.

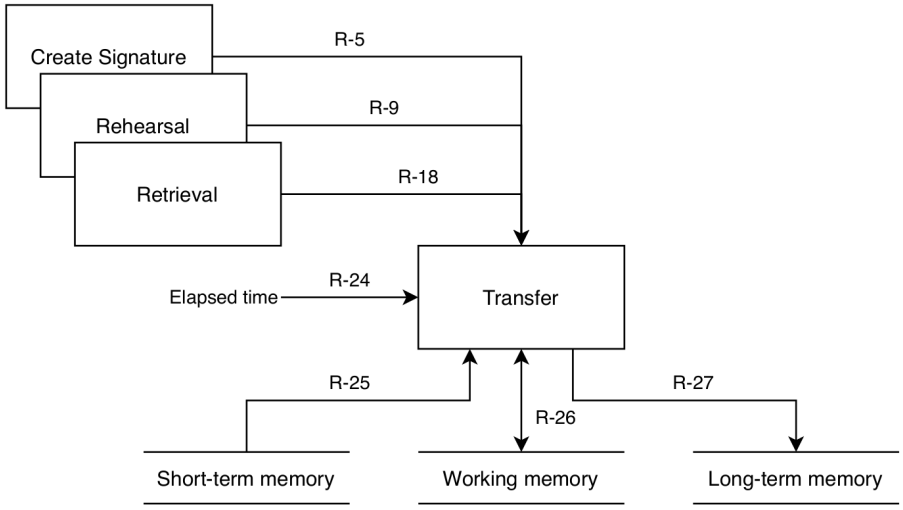


**Figure 2.9:** Dependencies present in the Transfer module (image from Mark te Brake (2021)).

| Dependency | Direction | Description |
|---|---|---|
| R-5 | In | Number of the newly added signature. |
| R-9 | In | Number of the deleted Signature. |
| R-18 | In | Amount of retrieved Signatures. List of Signatures that can not be transferred to LTM. |
| R-24 | In | Elapsed time since the start of the current iteration. |
| R-25 | In | Data of Signatures in STM for selecting a possible transfer. |
| R-26 | In | Data of Signatures in WM for selecting a possible transfer. |
| R-26 | Out | Delete Signatures transferred to LTM. |
| R-27 | Out | Store the transferred Signatures to LTM. |

**Table 2.7:** Description of the dependencies present in the Transfer module.

# 3 Analysis and design

By definition, modularity is a system property which measures the degree to which the components within a system can be decoupled and recombined. This concept is often utilized to break complex systems into functional blocks with different degrees of interdependence adding abstraction and flexibility to the system.

In SLAM, the concept of modularity is often applied to add independence between the sensor data processing and the SLAM. This allows the system to work with heterogeneous sensor arrangements, adding flexibility and easing the addition of new sensors and techniques.

## 3.1 Related work analysis

First of all, the solutions proposed by the different authors in the related work will be analyzed. This analysis will motivate the choices made for the design of the modular framework.

Jeroen Minnema (2020) is a modern proposal for modularity in EKF SLAM, graph-based SLAM and PF SLAM using a sensor categorization into idiothetic, absolute allothetic and relative allothetic. The main limitation of this technique is found when a sensor or technique does not fit the standards of the definitions, because then it can not be used by the framework. This is the case when using a LiDAR and a scan-matching technique like in Hess et al. (2016), which does not fit into the relative allothetic sensors' pipeline because it generates pose-to-pose constraints without using landmarks. Another example of this is the visual loop closure technique used in Labbe and Michaud (2013), which again is not based on landmark detection but instead uses a pose-to-pose constraint generation based on visual words matching, which makes it incompatible with the proposed design. On the other hand, this categorization of sensors may be unnecessary if other approaches are followed, like the ones seen in Blanco-Claraco (2019) and Colosi et al. (2020), which achieve modularity without using sensor categorization. For this project, techniques like LiDAR scan-matching and the loop closure detection used in RTAB-Map need to be supported in order to work with the state of the art open source solutions.

In Joost van Smoorenburg (2020), it is studied how to implement Minnema's design into RTAB-Map. In this project it is possible to see that the loop closure detection and loop closure constraint generation used by RTAB-Map does not fit any of the sensor categories and instead, in the conversion to Minnema's design, it is mapped into the SLAM algorithm block. The implementation is incomplete due to a lack of time, for this reason it is not clear how this problem would be addressed.

A distinct approach was found in Brake's work, where he tried to identify common processes found in different SLAM algorithms. The analysis of SLAM given in Mark te Brake (2021) is similar to the one given by Colosi, but focusing on the similarities found between RTAB-Map and ORB-SLAM2. The choice of these two SLAM solutions, makes the analysis very detailed but at the same time limits the scope to only visual SLAM algorithms. The set of components defined by Brake helps to understand common *blocks* found in both SLAM solutions, but further development needs to be made in order to work with other kind of sensors apart from the visual ones. Every sensor inputs a different type of data, and therefore every type of input source needs a different kind of processing before entering a block common between sensors, as described in Colosi et al. (2019). Furthermore, the definition of very specific components may benefit from abstraction, which would make more intuitive what components are necessary to build up a SLAM system.

In Colosi et al. (2020), the different components used in the most popular front-ends systems like in visual/LiDAR odometry or loop closure detection are analyzed, trying to implement a modular front-end capable of working with most of these systems. This framework consists

of a set of main modules that perform the most common tasks present in a front-end, like feature extraction and matching, and other sub-modules that complement the main ones and prepare them to work with different sensors. This design improves modularity adding support for different sensors, although it needs to implement new sub-modules every time a new sensor is added, so they are compatible with the framework.

In Blanco-Claraco (2019), the authors use the front-end and back-end definition given by Grisetti et al. (2010) to improve modularity in a graph-based SLAM system. In this work, Blanco proposes a back-end with a well defined interface to which several front-ends can be connected. Furthermore, Blanco also designs different kinds of front-ends like a stereo camera module and a LiDAR module which is capable of mapping and performing odometry.

Overall, there are many interesting approaches trying to add modularity to SLAM and they all add value towards this goal, but for this project we need one capable of working with RTAB-Map. For this reason, and for simplifying the scope of the project, only the solutions compatible with graph-based SLAM are taken into account (as RTAB-Map is a graph-based SLAM algorithm). Hereby, the work in Lynen et al. (2013) and in Tim Broenink (2016), presented in Section 1.3, contribute to how to devise a modular SLAM framework but their designs can not be directly applied for this project because they are limited to filtering methodologies. Furthermore, the solutions that support the largest number of types of sensors and techniques are preferred.

In order to clearly identify the differences in supported methodologies between the approaches, a comparison is given in Table 3.1 only taking into account the most used techniques in the state of the art. In this table, it is possible to see that Colosi's and Blanco's work are the ones that support the largest amount of sensor systems and techniques in RTAB-Map. Both of them use a front-end and back-end differentiation (explained in Section 2.2). During this thesis, instead of focusing on a modular front-end as in Colosi's work, we will focus on a back-end into which any front-end could be implemented. The main reason for this is the need of custom sub-modules in Colosi's model, but also there exist many open source front-ends available in which we could test our design but not that many standalone back-ends.

| **Author** | **Supported odometry techniques** | | **Supported loop closure detection techniques** | | | **Supported SLAM solutions** | | |
|---|---|---|---|---|---|---|---|---|
| | *Visual* | *LiDAR* | *Visual (BoVW)* | *LiDAR (Scan-matching)* | *Landmark-based* | *EKF* | *PF* | *Graph-based* |
| Minnema | x | x | | | x | x | x | x |
| Smooremburg | x | x | | | x | x | x | x |
| Brake | x | | x | | x | | | x |
| Colosi | x | x | x | x | x | | | x |
| Blanco | x | x | x | x | x | | | x |
| RTAB-Map | x | x | x | x | x | | | x |

**Table 3.1:** Comparison between the related work designs, only taking into account the most used sensor systems in SLAM. RTAB-Map's supported techniques are shown for comparison.

## 3.2 RTAB-Map modularity issues

RTAB-Map is a SLAM solution that has been developed for more than 10 years now (Labbé and Michaud (2011), Labbe and Michaud (2013), Labbe and Michaud (2014), Labbé and Michaud (2018), Labbé and Michaud (2019)) and has a big reputation among the scientific community in the sector. It started as a visual graph-based SLAM algorithm with a weight-based memory management system, and nowadays supports a great number of different front-ends and optimizers. Most of RTAB-Map's functional structure is available in its research papers and a brief review of the main components and their dependencies is given in Section 2.3. Nonetheless,

the code base of RTAB-Map lacks of official documentation and a big part of its functionality is mainly implemented in two large classes, which makes the reuse of code very complex.

As stated in the goals for this project (Section 1.4), one of the main objectives is improving the modularity in one of the main open-source SLAM frameworks, having selected RTAB-Map for this task because of its popularity and the available previous work with the framework in the RaM department. For this reason, in this section we will review the aspects of the architecture of RTAB-Map that affect its modularity. The following modularity issues are extracted from the work of Mark te Brake (2021), in which I paraphrase the outcomes of his project:

- **The STM and WM differentiation**: The differentiation between the STM and the WM highly increases the complexity of the code, generating a lot of dependencies between modules (R-3, R-4, R-6, R-7, R-10, R-14, R-19, R-22 and R-25, shown in Section 2.3.5). As it was previously analyzed in Mark te Brake (2021), the functionality added by this separation could be implemented in a different way merging the STM and the WM, reducing significantly the amount of dependencies. For instance, the *Rehearsal* method only needs to access the last two signatures added to memory (R-6), which could be also done without the existence of the STM. The pose-graph optimization does not distinguish between memories and optimizes all the signatures as a whole, so there is no need for duplicating the data access (currently performed by R-22 and R-23) and the dependency R-22 could be included in R-23. In contrast, other dependencies like R-10 and R-25 benefit from the existence of the STM (by excluding the most recent signatures for the loop closure detection and for the *transfer*), but the responsibility of differentiating between recently generated and older Signatures could be integrated into the loop closure detection and the transfer modules themselves. This modification should be properly designed to not alter the current code base affecting other functionalities, but it would significantly simplify the architecture of the framework.

- **The loop closure system**: Another functionality of RTAB-Map that affects its modularity is the loop closure detection and the loop closure constraint generation. In its origins, RTAB-Map was an appearance based system, and even thought it now supports additional sensors like LiDARs for position refinement and odometry (Labbé and Michaud (2018) and Labbé and Michaud (2019) respectively), the loop closure detection technique remains appearance-based since the beginning. This fact makes RTAB-Map dependent of visual data input (R-1), making compulsory the visual data stream. Moreover, there is no option for replacing the loop closure system and the framework does not provide with an interface for the input of additional loop closure constraints. This issue affects the framework reusability for platforms that do not mount a camera, or for robots that work in environments in which the camera it is not a reliable source of data for loop closure detections.

- **Dependencies in the memory management system**: The memory management system (the rehearsal, the transfer and the retrieval modules) uses signals that carry flags or counting values (dependencies R-4, R-5, R-8, R-9, R-17, R-18 and R-21) to exchange information about the creation or modification of Signatures. For example, it informs bout the generation of a new Signature, the number of Signatures transferred from one memory to another, or the need of triggering a graph optimization. These interfaces increase the number of dependencies between modules and makes code reusability more complex. To simplify the current architecture, the modules using these signals could extract the information by themselves by accessing the memory. Again, this modification should be properly designed to work with the current version of the framework.

  Another aspect of the memory management system that affects modularity is the dependency of the Rehearsal module to appearance data (as explained in Labbe and Michaud

(2013) and represented by R-6 in Section 2.3.5). This dependency appears with the use of the visual weighting system, that only works with visual input by comparing the number of matched visual words. The use of these weights create an indirect dependency in the other memory management modules, Transfer and Retrieval, that use this weighting system to decide which Signatures should remain in WM or which ones should be stored in LTM. This dependency to visual data affects the framework reusability in SLAM, because even though it supports other sensor input like LiDAR, the framework will reclaim the input of visual data by raising an error.

Furthermore, during the study of RTAB-Map another modularity issue regarding the pose-graph generation interface was found. The signature creation is the only documented module that accepts input for the graph generation. It only has two possible inputs: odometry and visual or LiDAR data. The odometry input is fine, because it allows the external generation of the odometry, contributing to modularity. But the problem is, that there is no input for loop closure or proximity constraints, which neglects the possibility of having external loop closure or proximity detection systems, and makes compulsory the usage of the ones integrated in RTAB-Map. This is a barrier for supporting other kinds of sensors, like the WiFi sensor system in Mathieu Nass (2020).

To summarize, RTAB-Map is not the best framework for being reused, because of its high number of dependencies and the absence of a well-defined interface. Most of the dependencies in RTAB-Map come from the first versions of the system that were dragged into the posterior iterations. Originally, RTAB-Map was a visual SLAM system and did not implement an interface for the substitution of major functionalities like the loop closure detection and constraint generation and the memory management system was dependent on its visual functionalities. These modularity issues remain in the current architecture of RTAB-Map. Ideally, a modular SLAM system would not depend on any specific sensor type and it would implement an interface that reduces the amount of unnecessary dependencies, easing the reusability of the different modules independently of the rest of the implementation. The modular SLAM framework proposal will achieve this by using self-contained modules and a well-defined interface.

## 3.3 Design motivation

In this section, the design choices and the motivation behind each decision is explained before showing the design proposal. As seen in Section 3.1, the architecture that better fits the structure of RTAB-Map is the one designed by Blanco. However, during the implementation phase, I used three weeks to implement Blanco's architecture with RTAB-Map without any success. The problems I encountered are: the complexity of adapting RTAB-Map's modules to the virtual classes and inheritance used in Blanco's framework, and the wide usage of the library MRPT of which Blanco is also the author, which increased the time needed to understand the code. For this reason, instead of directly using Blanco's design, a new design will be proposed using the concepts explained in his paper and the knowledge acquired from the other related work authors.

During the design of the new architecture, several trade-off needed to be made. First of all, the supported SLAM paradigms should be decided. The only related work that supported the three most common SLAM paradigms was the design proposed by Minnema. However, this design was not compatible with some of the functionalities of RTAB-Map. For this reason, the study of EKF and PF SLAM was left for further work.

Next, the degree of modularity of the framework needed to be established. Some of the related work, like the one by Brake, studied the modularity of the whole system, dividing into different modules the sensor processing components, and the SLAM algorithm. Others, like Colosi and Blanco focus more on one half of the system. In the analysis of Section 3.2, we could see that

most of the modularity issues come from the modules in the back-end and the definition of the interface for generating the pose-graph. For this reason, even thought the modularity degree of the front-end will be lower, we will limit the scope to the back-end and interface definitions, treating the sensor processing components as a black box.

Subsequently, the kind of map must be selected. What kind of information is stored into the map: topological only or with metric information, whether or not to store sensor data in the back-end and what kind of elements will be used for the pose-graph. Firstly, the elements of the pose-graph are decided. Being the ones described in Section 2.2, the selected ones. The reason for this is that the theoretical definition of the elements will be the closest to a standard. Regarding the other options, no metric nor sensor data will be stored in the back-end. The main reason is that they are not necessary to achieve the main goal of this project and they could easily be added in a later iteration of the framework. The same happens to the implementation of a memory management system.

## 3.4 Modular SLAM framework proposal

In this section, the design of the SLAM modular framework will be built upon the motivation showed in Section 3.3, solving at the same time some of the modularity issues found in Section 3.2. The framework proposal is sensor agnostic, and independent of the implementation of any front-end. It consists of a back-end, capable of communicating with the different front-ends to generate the pose-graph by using API calls. This kind of implementation makes the definition of the interfaces and its data flow very straight-forward.

### 3.4.1 General overview

A general overview of the framework's architecture can be seen in Figure 3.1. In this diagram the white blocks depict the modules that constitute the designed framework, and the black blocks represent the external implementations of the data collection and front-ends. The framework is mainly composed by the back-end and the interfaces to the front-ends. The back-end is capable of performing all the basic functionalities for SLAM, which are the graph generation and storage, optimization and visualization. The interface is the one that receives the calls from the front-end, stores the specific data from each front-end and takes care of the communication to the back-end.

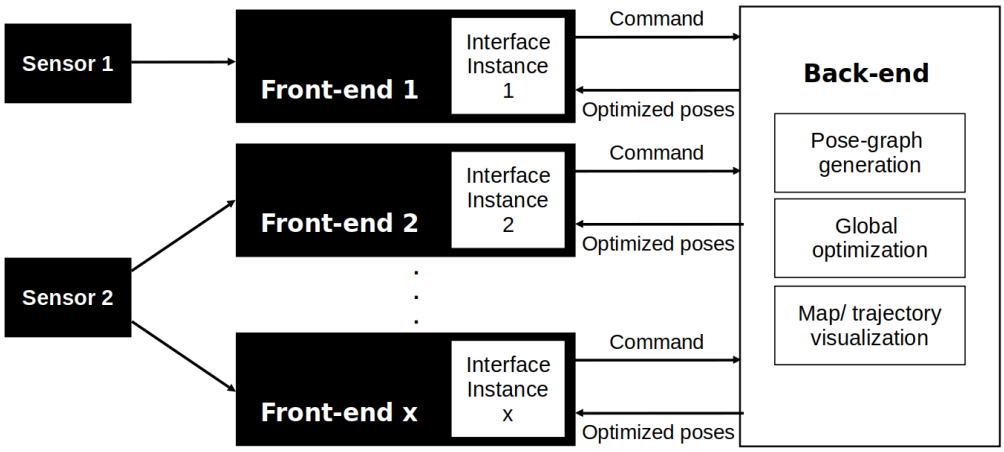


**Figure 3.1:** General overview of the framework's architecture. In white it is represented the "*blocks*" that constitute the framework designed in this thesis. The black modules represent the external implementations (sensor data collection and front-ends).

The framework is based on the suppositions:

- There is a single pose-graph with a unique global origin (relative coordinate frames will be expressed using this reference) formed by the defined entities and factors.
- The pose-graph forms an overdetermined system, meaning that there will be more constraints than nodes.

The developer is responsible of providing the framework with the right information. If both conditions are satisfied, the framework will generate the pose-graph, optimize it, and represent the trajectory. The framework it is also capable of providing the front-end with the optimized poses for more accurate estimations.

### 3.4.2 Front-end interface and communication with the back-end

The interface is a block designed to be integrated into the front-ends and is the one in charge of establishing the communication channel between the front-end and the back-end to allow the interaction. The interface object will establish the communication channel with the back-end as soon as the process is started. Once the communication channel is established, the interface will receive the requests from the front-end and send the commands to the back-end.

Every front-end will have its own interface. The interface not only sends the commands to the back-end, but also stores meta information from the front-ends. For instance, the interface stores information of the numbering of each graph element. Each front-end has its own numbering system, but they must be unified to the numbering system of the back-end, so this task is carried out by the interface. An example of this can be seen in Figure 3.2, where there are two front-ends connected to the framework sending information of landmarks detected in the environment. In this example, both front-ends use the same odometry information (nodes in white connected by arrows), but they detect different kind of landmarks (represented in red stars for the front-end 1 and green for the front-end 2). The interface connected to each front-end receives the landmark data (thick black arrow) and updates the IDs of the landmarks in a FIFO (First In First Out) fashion, maintaining a global coherence for the numbering. The landmarks with updated IDs are then sent to the back-end to add them to the pose-graph.

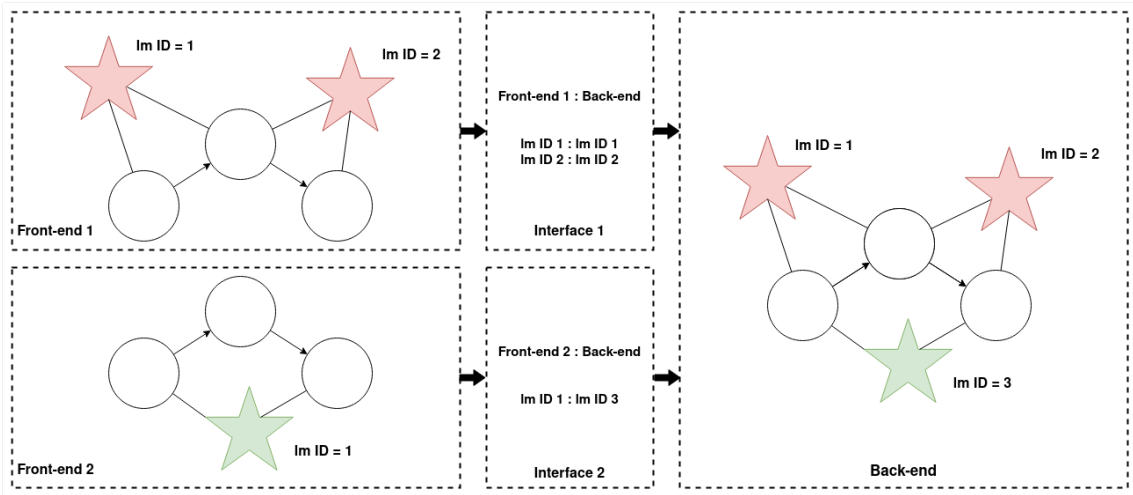


**Figure 3.2:** Interface numbering management example with landmarks.

### 3.4.3 Pose-graph generation

The back-end is designed to support the most essential functionalities in SLAM, being one of the most important ones the generation and storage of the pose-graph. The pose-graph generation module is the one that performs this task by receiving the commands sent by the interface.

The generated graph is composed by the three basic elements explained in Section 2.2: nodes, edges and landmarks. In the following lines it is explained how these elements are used to form the pose-graph used by our back-end.

- **Nodes**: Nodes are elements of the graph that represent spatial information, like poses of the robot in different time instants. These elements are identified by a unique ID. The spatial data stored by a node is in global coordinates, relative to a global reference frame (usually the initial pose {0,0,0}), and this data is used as an initial estimation during the pose-graph optimization.
- **Landmarks**: Landmarks are elements that represent unique features in the environment. They are related to the node in which they were detected, keeping track of all the nodes that observed the same landmark. These elements also make use of an ID. Landmarks are also used to trigger pose-graph optimizations when a known landmark is revisited.
- **Edges**: These elements represent spatial constraints between two nodes or between a node and a landmark. The spatial information stored by the edges is relative to the two connected elements of the graph. Together with the spatial information, the edges stores the uncertainty of the observation that generated it.

The pose-graph could use more information, like kinematic data to add redundancy and make the system more robust (Blanco-Claraco (2019)). Other authors studied the addition of different kinds of factors into the SLAM problem (Dellaert (2012)), but in order to test the core functionality of RTAB-Map we will only need to consider the most essential elements in graph-based SLAM.

Now that the elements that compose the pose-graph are known, it will be explained the available commands that the interface can send to the pose-graph module.

- *AddRefFrame()*: This command gets a pose as an input (i.e SE(2) or SE(3)). The pose is used to set the global reference frame of the pose-graph, which by default is set to {0,0,0}.
- *AddKeyFrame()*: This call adds a node to the pose-graph. It can receive as inputs: the ID and the pose information.
- *AddLandMark()*: This call adds a landmark entity to the pose-graph. It allows the input of the relative spatial information from the node to the detected landmark and the uncertainty related to the detection.
- *AddPoseConstraint()*: This command takes as inputs the data of the transformation between two nodes and the uncertainty related to it, adding an Edge to the graph using this information. Other kinds of constraints like range-only and bearing-only constraints are easily supported but not implemented for this project to limit the scope.
- *Optimize()*: Which manually triggers the optimization of the graph.
- *GetOptimizedPoses()*: Which returns the optimized poses of the graph.
- *SaveRawPoses()*: Which saves the graph into a file, so it can be latter used for the metrics and the plots.

### 3.4.4 Global optimization

The back-end also contains a pose-graph optimization module, which is an essential part of a SLAM framework. The optimizer is capable of accessing the pose-graph in memory to use the pose data during the optimization. The optimization is triggered once a loop closure constraint is added to the pose-graph, or in other words, when a new constraint is added between two non-consecutive nodes.

During the optimization, the error accumulated is reduced by using least-squares minimization (as explained in Section 2.2). To solve the minimization problem there are several algorithms available in the state of the art libraries. During this thesis, only one library will be implemented (as it will be explained in Section 4.1), but the optimization module is detached from the rest to easily implement other solutions.

### 3.4.5 Trajectory visualization

The framework also includes a visualization tool. This visualizer serves for debugging, as it plots the trajectory in real-time, but it also serves for testing. The framework can save the generated pose-graph into a file for its use in the evaluation step.

# 4 Implementation

In this chapter, it is explained the implementation of the designed modular framework and its integration with other open-source solutions. There are several possible languages in which the framework could be implemented, but for this project C++ was chosen, for its versatility and because most of the open-source solutions use this language in their code base (including RTAB-Map), avoiding possible incompatibilities. The framework is designed as a stand-alone library which can be installed using CMake to ease the import in different projects.

## 4.1 System overview

As seen in Section 3.4, the modular framework consists of an interface that receives the information from a front-end and a back-end that solves the SLAM problem with this information. An overview of the implementation of the system can be seen in Figure 4.1.

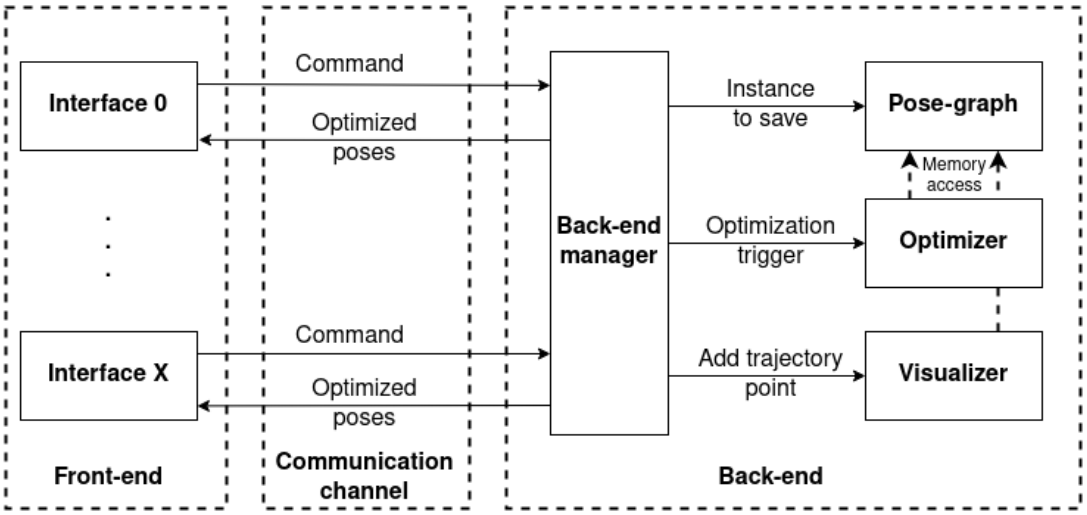


**Figure 4.1:** Implementation of the modular framework.

### 4.1.1 The interface and the communciation channel

The interface must be implemented into every front-end that the user is willing to use, by including the *WorldModelInterface.h* file and creating an instance of the class.

The interface supports the input and output of data in different formats, which are commonly used by the state of the art. For instance, the spatial information can be received and given as an output in the form of quaternions, matrix form or with Euler angles.

Once the interface is connected, it will take care of establishing the communication channel with the back-end, and the front-end will be able to perform the API requests. For this project, two communication protocols were implemented. The first one, the framework is implemented as a single process, and a zero copy protocol was used, sending the information with pointers. In the second one, a inter-process communication is allowed using a UDP server-client communication protocol.

With the first communication protocol there is only one instance of the back-end and this is coded as a *singleton* (a class that can only be instantiated once). This means that when an interface instance is created, the interface looks for the instance of the back-end and stores its

address to communicate with it. For this reason, all the implemented modules must be part of the same process.

The second communication protocol allows for an inter-process communication. This protocol allows an easier implementation of different front-ends, because they do not need to be integrated into the same program, having a better separation of concerns and a faster configuration of the project. In this case, the interface behaves as a client and the back-end as a server. The data-frames contain the same information of the commands.

### 4.1.2 The back-end

The modularity of the back-end resides in the definition of the modules, which are based on the theoretical definitions of each functional block. Each module is defined to be self-contained and only allow the necessary data flow. As seen in Figure 4.1, only two dependencies could be eliminated: the "optimization trigger" and the "add trajectory point" flags. These dependencies could be replaced by adding the functionality of generating the flags inside of each concerned module, but were generated to ease the implementation of the code. As explained in Section 3.3, the map only contains topological information. For this reason, the optimizer and the visualizer modules only have access to the spatial data. If any of the modules were to be replaced or modified, the exposed dependencies should be taken into account keeping the same data flow. In a further iteration of the framework, it is possible to define abstract classes that take care of these interface definitions to increase the degree of modularity.

The back-end is composed by three main modules that are managed by the back-end manager: the pose-graph, the optimizer and the visualization tool.

The back-end manager is the one that receives all the commands from the interface and creates the necessary object instances or calls the other modules' functions, similar to how a factory does it. Without the two aforementioned flag dependencies, this module could be renamed as the pose-graph generation module.

The pose-graph module works as a memory pool, in which all the instances created by the back-end manager (which currently are: nodes, constraints and landmarks) are stored.

The optimization module performs the pose-graph optimization whenever the back-end manager triggers the optimization. The optimizer accesses the memory address of the pose-graph, and translates the information into the desired optimization library. The optimizer is currently implemented using GTSAM (Dellaert (2012)), but others like g2o or Toro could also be implemented using the current implementation as a reference. Once the optimization is calculated, the optimizer updates the pose-graph with the new values.

The visualizer is in charge of showing the trajectory in real-time. It is implemented using SFML (Laurent Gomila (2021)), and it is in charge of showing a plot that is updated with commands from the back-end manager. This module is an adaptation of a community plotting tool with SFML called "SFPlot" (SFPlot (2021)) and currently serves as a debugging tool, so its implementation it is not meant for modularity and may not work in every scenario.

## 4.2 Implementation with RTAB-Map

A new framework is very unlikely to be used if other, more popular options, exist. The best way to let others to know about a new framework, when there are so many already, is to prove its usefulness in another well-known software. For this reason, it is important to prove the concept by combining the framework with RTAB-Map, which already has a huge community.

However, because of RTAB-Map's numerous dependencies between modules (as seen in Section 3.2), this is no easy task. To implement the proposed framework into RTAB-Map, we must identify which modules of RTAB-Map will be substituted by our implementation. In Table 4.1

and Table 4.2 it is showed a categorization of the different modules into the front-end and the back-end.

| FRONT-END | |
|---|---|
| **Module** | **Description** |
| RTAB-Map Odometry (VO, LO, check Section 2.3) | Odometry generation. |
| Rehearsal | Key-frame generation. |
| Bayes filter + constraint generation | Loop closure detection and constraint generation. |
| BoVW | Feature extraction and storage for the loop closure system. |

**Table 4.1:** RTAB-Map modules mapped into the front-end.

| BACK-END | |
|---|---|
| **Module** | **Description** |
| WM | Holds the pose-graph. |
| Optimization | Pose-graph optimization. |
| Retrieval + Transfer + LTM | Memory management system. |

**Table 4.2:** RTAB-Map modules mapped into back-end.

Ideally, the back-end of our modular SLAM framework would directly receive the information from the RTAB-Map modules mapped to the front-end using the defined commands. In addition, the back-end would also replace all the RTAB-Map modules categorized as back-end. This replacement, together with the defined interface would solve most of the modularity issues of Section 3.2.

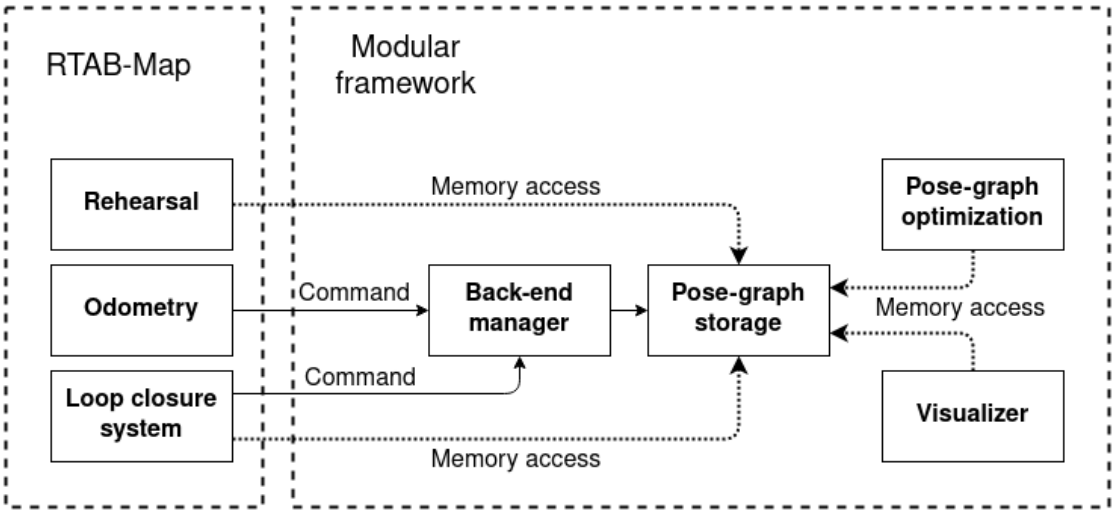


**Figure 4.2:** Modular SLAM framework ideal implementation with RTAB-Map.

Nevertheless, the full integration could not be implemented due to the time limitation. RTAB-Map has a huge code base that can not be reverse engineered in the available time for the project.

Instead, the framework was implemented as shown in Figure 4.3. In this figure, the dependencies in RTAB-Map are simplified for clearness, for a complete review of them please refer to Section 2.3.5. The modules marked in red are the ones that would be replaces by our back-end if a full integration were possible. Note that the global optimization module of RTAB-Map is disabled while working with our framework, because there is no need for having two optimizers working at the same time. The interface is hooked to the STM to receive the data from the Signatures after the Rehearsal and the loop closure detection. This is possible because the re-

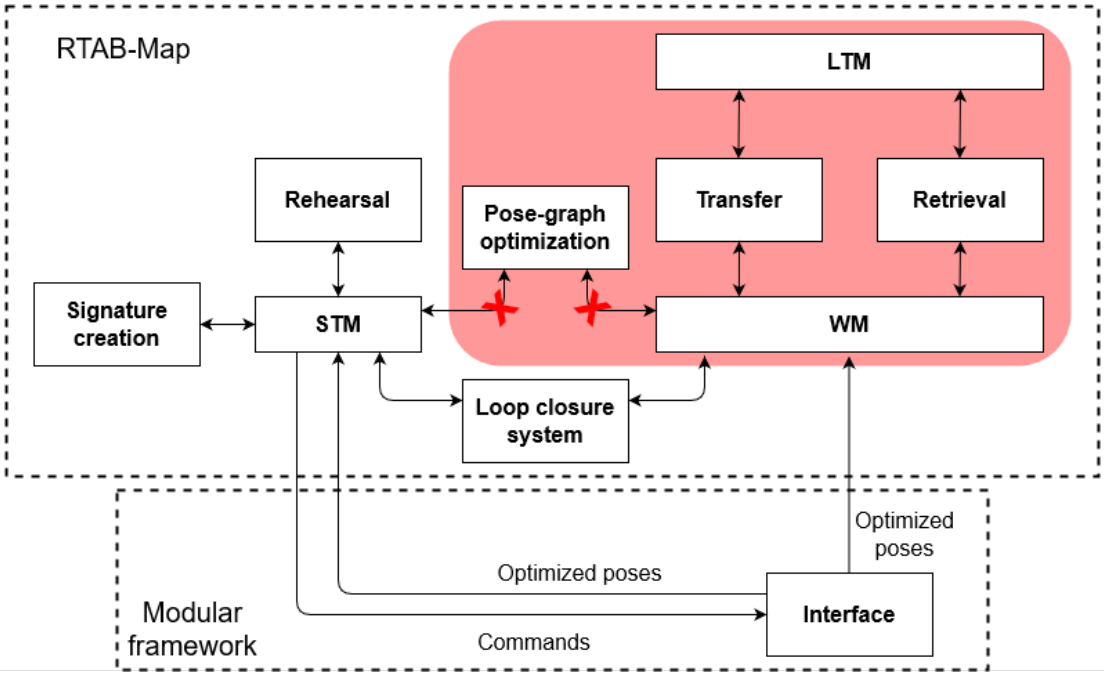


**Figure 4.3:** Modular SLAM framework implementation with RTAB-Map. The modules marked in red are the ones that would be replaces by our back-end if a full integration was possible. The red crosses in the pose-graph optimization module mark that it is disabled when working with our modular framework.

hearsal only takes into consideration the two most recent Signatures (as analyzed in Section 3.2), and the loop closure the most recent one, so there is no need for waiting for the Signatures to enter the WM. Furthermore, whenever there is a loop closure, the global optimization is triggered and the optimized poses are updated into both the WM and the STM.

All the modifications to RTAB-Map are made in its main file "rtabmap.cpp", and to the project configuration cmake file to import our library. The "rtabmap.cpp" file contains all the main workflow in RTAB-Map. The pipeline of a single iteration of the modified workflow of RTAB-Map would look as follows:

1. RTAB-Map receives the input data and creates a Signature.
2. It performs "rehearsal" over the new Signature and computes metric information using the input data.
3. The Bayes filter is updated with the new Signature data.
4. If the current Signature is a good candidate for a loop closure, the near nodes are retrieved from the LTM.
5. RTAB-Map looks for landmarks, proximity detections and loop closures adding any of them if they are detected.
6. At this point, the current Signature already contains all the information generated by RTAB-Map. Here is where the calls to our framework are made (as shown in Listing 4.1), using the commands to generate the desired instances in the pose-graph.
7. Optimization is triggered. In this case the call for the optimization is commented out and instead the optimize() command of our framework is used as seen in Listing 4.2. After the

opPoses are received, some "glue code" is necessary to translate the data into the format used in RTAB-Map.

8. RTAB-Map evaluates the Signatures for "Transfer".

```
anloro::WorldModelInterface anloroInterface;
anloro::Transform transform = anloro::Transform(x,y,z,roll,pitch,yaw);
anloro::Uncertainty uncertainty = anloro::Uncertainty(ux,uy,uz,uroll,upitch,uyaw);
anloroInterface.AddKeyFrame(id,transform);
anloroInterface.AddPoseConstraint(idFrom,idTo,transform,uncertainty);
```

**Listing 4.1:** Code used to do the API requests from RTAB-Map.

```
anloroInterface.Optimize();
opPoses = anloroInterface.GetOptimizedPoses();
```

**Listing 4.2:** Code used to obtain the optimized poses from our framework.

It is important to notice that "anloro" is the namespace used in our framework. In addition, the input data used in Listing 4.1 was previously obtained using the *getters* of the current signature and the "opPoses" in Listing 4.2 are transformed into the custom Transform class in RTAB-Map similarly on how ours works in Listing 4.1. This transformation is performed to not alter the regular work flow of the pipeline, and allows us to simply replace their optimization module commenting their call and adding the ones of our Framework.

## 4.3 Implementation with additional front-ends using ROS

The implementation with RTAB-Map serves as a proof of concept that it is possible to solve the SLAM problem without most of the dependencies that created the modularity issues in the original version. In addition, it shows that the designed modular framework is capable of working together with the wide variety of sensor processing modules present in RTAB-Map. However, from a functional point of view, the designed framework does not add any additional features to RTAB-Map apart from potentially reducing the number of dependencies. For this reason, to fully show the potential of the designed framework, we will implement additional front-ends to show that it is possible to easily add new functionalities by having several front-ends working together with our framework.

First of all, we will add an interface for ROS Noetic (ROS (2021)). The reason for this is that during this project it is not possible to reverse engineer the architecture of another big framework as it was done with RTAB-Map. Instead we will use the available ROS nodes to implement another front-end. In addition, ROS has a huge community of developers and robotics enthusiasts, it has support for a great variety of sensors and its architecture is modular and has well defined interfaces.

Two more front-ends are added: An Apriltag landmark detector (called apriltag_ros from the authors Wang and Olson (2016), Brommer et al. (2018) and Malyuta et al. (2019)) and an odometry node. To implement these new front-ends using ROS it is necessary to implement the interface into a ROS node that subscribes to the topics of the aforementioned modules. The subscriber will serve as the middleware between the selected modules and our interface, receiving the messages whenever they are published to the topic, unwrapping the ROS messages and transforming the data into a supported format. The described behaviour it is shown in Figure 4.4.

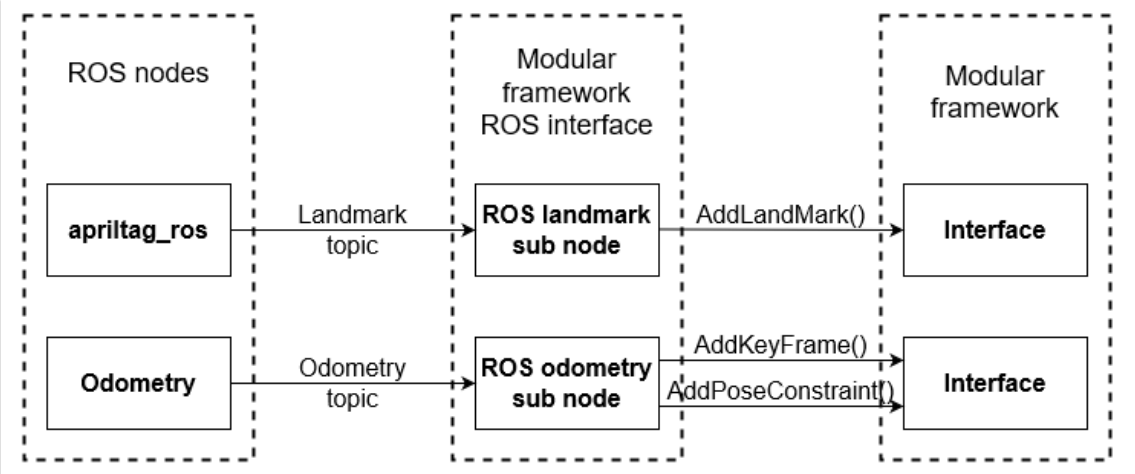


**Figure 4.4:** Modular SLAM framework implementation with ROS and the apriltag and odometry nodes. The modular framework's ROS interface subscribes to the necessary topics and adjusts the data format for the interface.

# 5 Evaluation methodology and results

This chapter presents the evaluation methodology and its corresponding results:

- To validate the proposed modular SLAM framework design.
- To obtain results that support the answers to the research questions.

Firstly, the experimental setup is described, including the hardware, models and methodology used. Afterwards, we detail the experiments designed to test the various front-ends and its combinations. Finally, the results of the different experiments are discussed.

## 5.1 Experimental setup

Due to the situation created by the pandemic of the Covid-19, the experiments will only be conducted in a simulated environment. The processing power used consists of a MSI G60 laptop with an Intel Core i7-4720HQ CPU at 2.6GHz with 8 cores, 16 GB of working memory and a Nvidia GTX 960M GPU. The laptop runs on Ubuntu 20.04 (Focal Fossa) and uses ROS Noetic. The selected simulator is the ROS-based Gazebo simulator in its 11th version (gaz (2020)). Gazebo is capable of simulating 3D environments and has numerous plugins available, for example, the differential drive plugin is capable of receiving velocity commands and output odometry information, which is very useful to simulate a tele-operated robot.

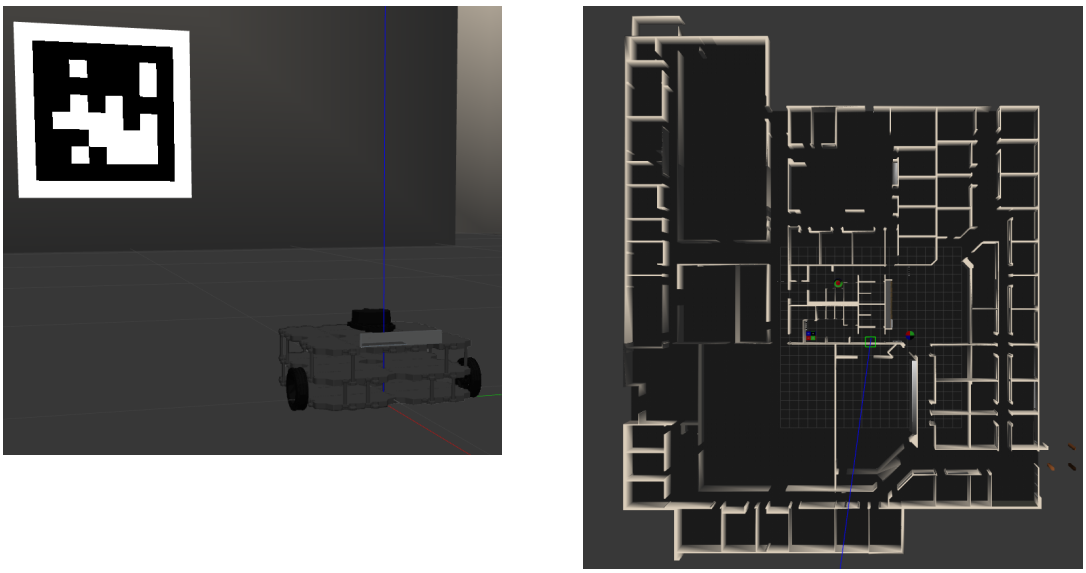

**Figure 5.1:** (Left) Turtlebot3 Waffle model with April tag model in the background. (Right) Willow garage model seen from the top.

For the simulation, we need a robot and an environment model. As the modeling of an specific robot is out of the scope of this thesis, we will use the model of a commercial robot. The one selected for this task is the Turtlebot3 in its Waffle version (ROBOTIS (2021)), which 3D model, kinematic model and libraries are available as open-source. The Turtlebot3 Waffle is a two-wheeled differential drive robot that is equipped with wheel encoders, an IMU and mounts a LDS-01 LiDAR and an Intel RealSense R200 RGB-D camera. The variety of sensor used in the Turtlebot3 makes it polyvalent and a great candidate for the experimentation in this project. For the environment, we selected the well-known Willow garage model, which has a good size and it can be modified to add any necessary element. Additionally, some textured objects were

included in the model to ease the visual loop closure detection. In order to use the landmark detector, we also need the April tag 3D models and they must be added to the Willow garage model. There are several models available online, so we used the ones provided in "april tags (2019)" and manually add them to different spots of the environment. The different models can be seen in Figure 5.1.

All the different experiments will be carried out in a *passive SLAM* fashion, which means that the robot will not perform path planning and it will be controlled remotely using the keyboard with Gazebo's differential drive plugin. The robot will be driven through different predefined paths and all the sensor data necessary for the algorithms will be stored into "*rosbags*", which is a format used in ROS, designed for storing sensor data. Using the same *rosbags* will help us in developing a comparative analysis between methods.

## 5.2 Experiments

To validate the benefits of using the designed modular framework, three different experiments were created. The aim of these experiments is to validate the compatibility of the framework with different sensor types and the capability of receiving information from them simultaneously while solving the SLAM problem.

As mentioned in the previous section, all of the experiments will be carried out using the same data. The ground-truth and the drifted odometry that will be used as input for the different experiments can be seen in the Figure 5.2.

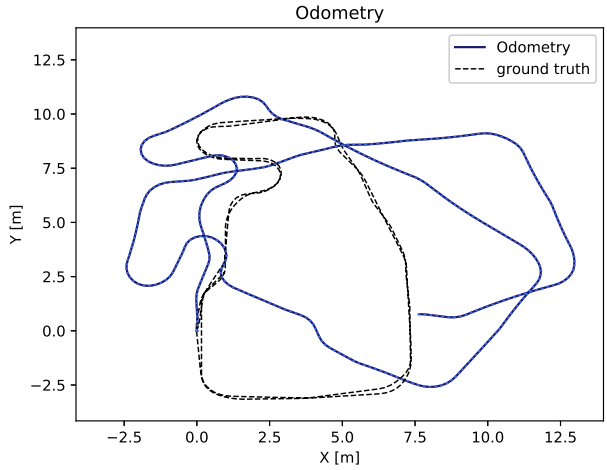


**Figure 5.2:** Ground truth (in black) and drifted odometry (in blue).

### 5.2.1 Experiment 1: RtabMap with the designed framework.

***Goal**: Evaluate the compatibility of the designed modular SLAM framework with RTAB-Map.*

The first experiment will test the implementation of RTAB-Map into our framework (explained in Section 4.2). RTAB-Map will receive the drifted odometry data and will make use of its visual loop closure algorithm. Both the odometry and the loop closure constraints generated by RTAB-Map are fed to the back-end, which will optimize the trajectory and return the optimized poses to RTAB-Map to improve the quality of the estimations. The resulting graph will be stored and plotted for its comparison with the rest of the results.

**Validation**: To validate the results, the estimation obtained by the combination of RTAB-Map and the designed framework will be compared with the ground truth. The results will be evaluated using the metrics in Section 5.3. To set a reference, the drifted odometry and the estimation with the original version of RTAB-Map using GTSAM will be also compared with the

ground truth using the same metrics. If the designed framework is compatible with RTAB-Map, the expected result should be similar to the ones obtained by the original version of RTAB-Map without any modifications.

### 5.2.2 Experiment 2: Odometry and Apriltag detector nodes with the designed framework.

***Goal**: Test the proposed design as a stand-alone SLAM framework and its capability of supporting simultaneously the input from different front-end modules.*

The second experiment will test the usage of the proposed design as a stand-alone framework using other front-ends different from RTAB-Map. The designed modular framework is meant to work with any kind of constraints and not only with the ones created with RTAB-Map. For this reason, a complete SLAM framework is built using the implementation in Section 4.3. In this case, the same drifted odometry will be used, but this time the loop closure constraints will be added by the Apriltag detector, which will serve for minimizing the error introduced by the odometry. In this case, both the trajectory and the landmarks will be plotted.

**Validation**: The results will be validated in the same way as in experiment 1, using the metrics of Section 5.3.

### 5.2.3 Experiment 3: RtabMap and Apriltag detector nodes with the designed framework.

***Goal**: Evaluate usage of additional front-ends together with RTAB-Map.*

The third experiment will evaluate the addition of new constraints to the implementation with RTAB-Map. The more information is used the better the estimation. For this reason, if the information added by the Apriltag detector is sufficiently reliable, the estimated trajectory should be better than the one obtained with less data. This experiment will test the interface of the designed framework, and its usefulness to combine the data obtained with state of the art solutions like RTAB-Map and additional front-ends, like the Apriltag detector.

**Validation**: The results will be validated using the same metrics than in the previous experiments. In this case, the results obtained in this experiment will be also compared with the reference and the other two experiments. The expected result is a improvement in the estimation with respect to the other two experiments.

## 5.3 Metrics

In this section, the metrics used in the experiments will be explained. All the trajectory estimations obtained by the different configurations in the three experiments are compared with the ground-truth. Before the comparison, linear interpolation is used. The timestamp is used for interpolating the ground-truth so it coincides with the timestamps of the estimated poses. This way, the error due to the displacement during the time difference is minimized.

After the interpolation, the metrics of the Absolute Trajectory Error (ATE) and the Relative Pose Error (RPE) are computed. These metrics are shown below as described in the work of Sturm et al. (2012).

The ATE is computed as the Root Mean Squared Error (RMSE) of the absolute error $F$:

$$\mathrm{RMSE}\left(\mathbf{F}_{1:n}\right) := \left(\frac{1}{n}\sum_{i=1}^{n}\left\|\mathrm{trans}\left(\mathbf{F}_i\right)\right\|^2\right)^{1/2} \tag{5.1}$$

Where n is the total number of poses, $trans$ specifies the translational component of $F$ and $F_i$ is the absolute error at every time instant $i$:

$$\mathbf{F}_i := \mathbf{Q}_i^{-1}\mathbf{S}\mathbf{P}_i \tag{5.2}$$

Where $Q$ is the ground-truth, $P$ the pose estimation and $S$ the Horn transform. The ATE can be seen as a measure of the global consistency of the map. In contrast, the RPE can be described as the accumulated drift in a certain time interval $\Delta$. The RPE is computed as the RMSE of the relative error $E$:

$$\mathrm{RMSE}\left(\mathbf{E}_{1:n},\Delta\right) := \left(\frac{1}{m}\sum_{i=1}^{m}\left\|\mathrm{trans}\left(\mathbf{E}_i\right)\right\|^2\right)^{1/2} \tag{5.3}$$

Where $m = n - \Delta$, and $E$:

$$\mathbf{E}_i := \left(\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\Delta}\right)^{-1}\left(\mathbf{P}_i^{-1}\mathbf{P}_{i+\Delta}\right) \tag{5.4}$$

The RPE can be normalized over all possible time instants, obtaining a similar measurement that with the ATE. However, we will use the time interval $\Delta = 1$ to get the drift accumulated every time instant.

## 5.4 Results

In this section it shown the results obtained for the 3 experiments. There is a specific subsection for every experiment and final comparison between the all the experiment results and the results with RTAB-Map and the drifted odometry at the end.

### 5.4.1 Experiment 1: RtabMap with the designed framework.

As seen in Figure 5.3 and Table 5.1, the estimation obtained in experiment 1 is close to the ground-truth, but still there is some error that makes the estimation to tilt to the left. A possible reason for this is the usage of the default parameters given by RTAB-Map. The experiment is carried out in a simulated environment with very few visual features, and even though textured models were implemented in the map, some parameter tuning may be necessary for obtaining better results. Nonetheless, it is important to note that in this experiment we do not aim for improving the accuracy of the estimation, but to obtain similar results to the ones with the original version of RTAB-Map. This will be evaluated in the Subsection 5.4.4.

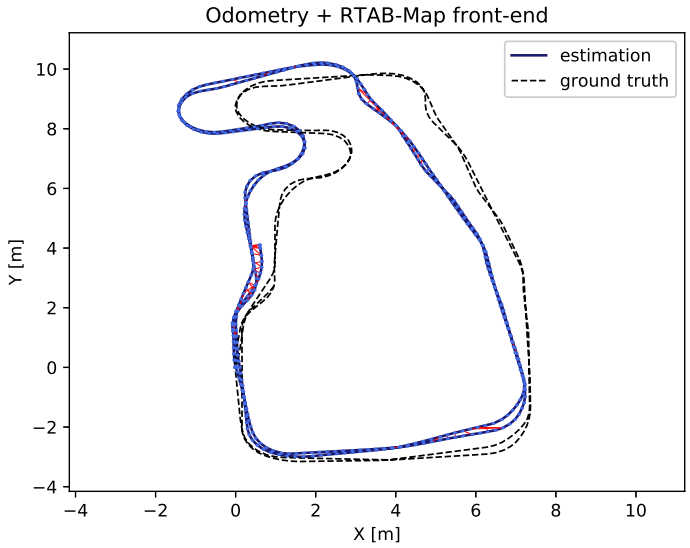


**Figure 5.3:** Experiment 1 results. The loop closure constraints are marked in red.

| Metric | Experiment 1 estimation |
|---|---|
| ATE (m) | 0.76 |
| RPE (m) | 0.05 |

**Table 5.1:** Experiment 1 metrics.

### 5.4.2 Experiment 2: Odometry and Apriltag detector nodes with the designed framework.

As it can be seen in Figure 5.4 and Table 5.2, the results of the experiment 2 are slightly better than the ones with experiment 1. In this experiment, the framework worked as a stand-alone SLAM framework, which proves that it is capable to work without RTAB-Map.

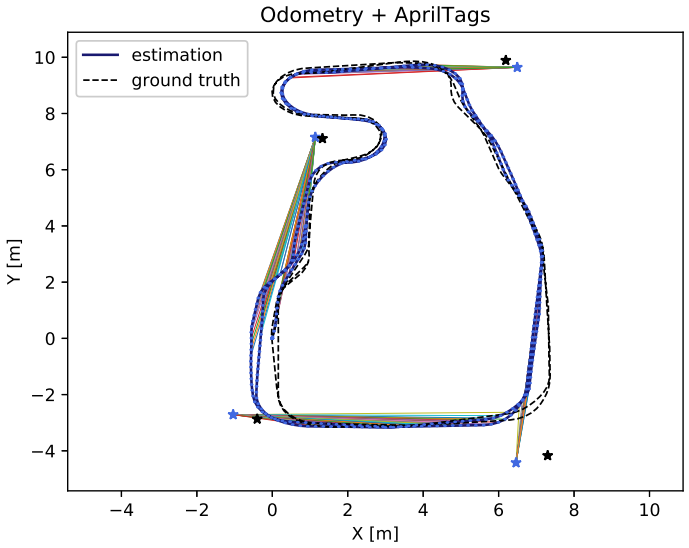


**Figure 5.4:** Experiment 2 results. The landmarks are represented as stars (blue for the estimation and black for the ground-truth). The constraints for the landmarks are represented as thin lines in different colours.

| Metric | Experiment 2 estimation |
|---|---|
| ATE (m) | 0.32 |
| RPE (m) | 0.05 |

**Table 5.2:** Experiment 2 metrics.

### 5.4.3 Experiment 3: RtabMap and Apriltag detector nodes with the designed framework.

The results of experiment 3 can be seen in Figure 5.5 and Table 5.3. Compared to the results of the previous experiments, it performs better than experiment 1, but worse than experiment 2.

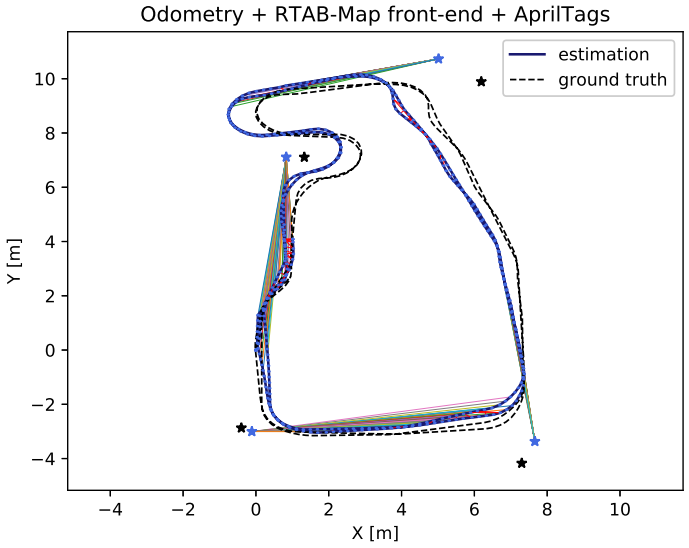


**Figure 5.5:** Experiment 3 results. The loop closure constraints are marked in red. The landmarks are represented as stars (blue for the estimation and black for the ground-truth). The constraints for the landmarks are represented as thin lines in different colours.

| Metric | Experiment 3 estimation |
|---|---|
| ATE (m) | 0.45 |
| RPE (m) | 0.05 |

**Table 5.3:** Experiment 3 metrics.

During the experiment, the uncertainty of each kind of constraint was observed. The uncertainty of the loop closure constraints given by RTAB-Map was lower than the uncertainty of the landmark detections. For this reason, it is possible the former had a greater impact on the final result than the latter. However the landmark constraints still have an influence in the final result reducing the global error and improving the results of Experiment 1. It is possible that a tuning of RTAB-Map's loop closure generation parameters would have led to better results. Unfortunately, the time limitation only allowed a testing with the default parameters. On the other hand, the designed framework it is not meant to verify the input data, which in this project is left as a responsibility to the user, as mentioned in Section 3.4.

### 5.4.4 Comparison of all the results with the original version of RTAB-Map.

In this section we compare the results of all the experiments and the results obtained with the original version of RTAB-Map using GTSAM and the drifted odometry, as can be seen in Figure 5.6 and Table 5.4. The results of the original version of RTAB-Map are very similar to the ones obtained in Experiment 1, with slight variances that may be due to the different implementations of the optimizer. This confirms the fact that the framework is capable of working with RTAB-Map, obtaining similar results to the original version. In addition, the framework proves to be capable of working as a standalone SLAM algorithm and support different kinds of input data from different sensors and combine them.

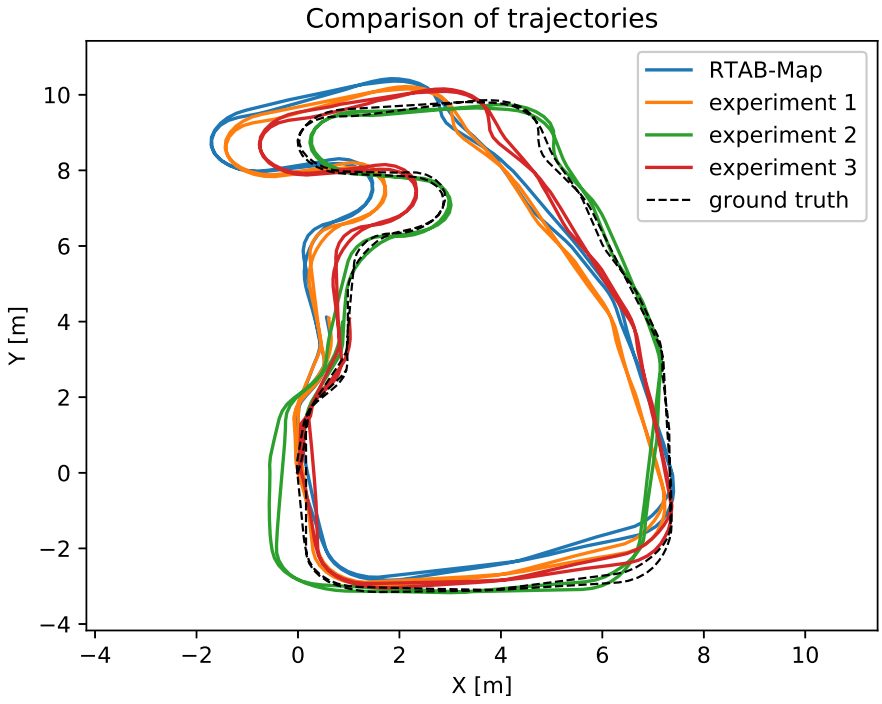


**Figure 5.6:** Trajectory comparison.

| Metric | Drifted odometry | RTAB-Map (GTSAM) estimation | Experiment 1 estimation | Experiment 2 estimation | Experiment 3 estimation |
|---|---|---|---|---|---|
| ATE (m) | 4.59 | 0.94 | 0.76 | 0.32 | 0.45 |
| RPE (m) | 0.15 | 0.05 | 0.05 | 0.05 | 0.05 |

**Table 5.4:** Metrics comparison.

# 6 Conclusion

## 6.1 Findings

In this section, the research questions are discussed by using the findings that contribute to their answers.

### 6.1.1 What are important aspects of modularity? How is modularity applied to SLAM?

This question is targeted at establishing the baseline of modularity concepts applied to SLAM, which is used for the whole project. Modularity in general is tackled in Section 2.1, which is directly extracted from the theoretical concepts of software reusability widely used in computer science. On the other hand, its application to SLAM is constructed from the study of the related work of previous authors, shown in Section 1.3.

As a summary, modularity and software reusability consists in the usage of architectures that promote the creation of independent modules and their interchangeability. This is achieved by creating self-contained modules with well-defined interfaces that reduce the number of dependencies between modules and the usage of different levels of abstraction.

Modularity applied to SLAM, is built upon the general concepts of modularity and software reusability, but with a strong focus in the practical matters. In modern SLAM solutions, modularity is applied to add independence between the sensor data processing algorithms and the SLAM solutions. The goal is to integrate an arbitrary number of sensors of different types into a unified SLAM solution. The creation of a modular SLAM system would significantly ease the task of reproducing research results and drastically reduce the time to market. The difficulty presented by this challenge is the implicit assumptions of each SLAM solution, regarding the robot sensory system, the robot model or the environment conditions.

### 6.1.2 What aspects of the current implementation of RTAB-Map are not compatible with modularity? How could these be modified to increase the reusability of the framework?

RTAB-Map is known for its various functionalities, but not for being the best framework to reuse its code base or add new features. In Section 2.3.5, it was shown an analysis of the numerous dependencies present in the framework. Most of them come from the origins of the framework, in which it was designed as a visual-based SLAM solution. These dependencies lead to the modularity issues shown in Section 3.2, and severely affect the reusability of the framework. Nevertheless, solving each modularity issue independently by directly modifying the related modules is a very time consuming and difficult task if you are not the author of the code. For this reason, a more time efficient and intuitive alternative is given.

Using the front-end and back-end definitions from Grisetti et al. (2010), and the study of the dependencies in Section 3.2, we can see that most of the modularity issues of RTAB-Map come from the modules that compose the back-end, and the definition of the interface for generating the graph. For this reason, to increase the reusability of RTAB-Map, we look for a new back-end architecture that satisfies the modularity aspects studied in the research question 1. This new architecture has to fit in the structure of RTAB-Map and maintain its core functionalities. The specifics of this architecture are analyzed in the third research question.

### 6.1.3 Which modular framework in the state of the art would best fit the structure of RTAB-Map?

The last question arises the need to find a framework compatible with RTAB-Map, and capable of using its sensor integrations. To find the most suitable framework, the main functionality of RTAB-Map was analyzed in Section 2.3, and compared with the state of the art. The outcome of the analysis determined the framework proposed by Blanco as the most suitable for this task (as seen in Section 3.1). However, as explained in the same section, the implementation given by the author could not be directly used for this project. Instead, the concepts transmitted in Blanco's paper, in combination with the knowledge gained with the other authors, were used to design a new framework compatible with RTAB-Map.

The designed framework is basically composed by an interface that communicates with each front-end, and a back-end that generates the pose-graph and optimizes it, while satisfying the fundamentals in research question 1. The detailed description of the design is shown in Section 3.4. The framework was implemented with RTAB-Map by converting the "*Signatures*" and "*Links*" into commands of the framework and substituting the optimizer by our implementation, as it was described in Section 4.2. The designed framework was validated using the methodology described in the experiment 1, which results show that the design is compatible with RTAB-Map, obtaining similar results to the original version.

### 6.1.4 What interface would allow the implementation of new sensors and algorithms into the framework?

A modular SLAM framework must be compatible with more types of constraints and data than the one provided by RTAB-Map. A general definition for the elements in a pose-graph, like nodes and constraints, is found in the bibliography and summarized in Section 2.2. These definitions were used in the creation of the interface of the designed framework, establishing a series of API calls for its usage in the different front-ends.

The interface is also implemented with different state of the art modules using ROS. This implementation includes an odometry module and an Apriltag detector which integration is detailed in Section 4.3. To test the interface and the capability of using different front-ends at the same time, test 2 was designed. The results of experiment 2, together with the results of experiment 1 demonstrate that the designed interface is compatible with different algorithms independently of their implementation or the type of sensor used. Additionally, the results of experiment 3 show that the proposal is also capable to work simultaneously with different inputs and use all the provided data in the estimation.

## 6.2 Discussion

The implementation of the new framework has several advantages due to its modular design. Nonetheless, because of the limited time, this framework does not have the more advanced features present in algorithms with a longer development time, like the ones present in the state of the art solutions. In this section, to improve the critical assessment of the work, it will be summarized the strengths and weaknesses of the current proposal.

**Strengths**

- *Compatible with RTAB-Map:* The main strength of the framework is its proven compatibility with RTAB-Map. This framework significantly contributes to achieving a fully modular version of RTAB-Map, which will significantly ease the development of new technologies with SLAM.
- *Self-contained modular back-end:* The back-end is independent of any front-end implementation, reducing the number of dependencies and easing reusability. In addition, the designed modules are very close to their theoretical definition, which eases the standardization of the interfaces.
- *Well-defined interface:* The interface provides with a series of API calls for the creation of the pose-graph using a format compatible with different open-source SLAM modules. The framework was integrated and tested with RTAB-Map and different ROS modules, demonstrating its compatibility.
- *Support for an arbitrary number of sensors of different types:* The framework provides the tools to solve the SLAM problem with any kind of sensory system capable of providing with a pose constraint and uncertainty of the measurement, which is the case for the vast majority of sensors used in SLAM.
- *Support for multiple inputs simultaneously:* The framework is capable of receiving data from multiple sources simultaneously, allowing the combination of different algorithms independently of their implementation. In addition, its compatibility with state of the art solutions opens the possibility of easily extending the functionality of algorithms like RTAB-Map.

**Weaknesses**

- *No time synchronization:* In the current version there is no time synchronization for the input data. For the experiments and the modules used in this project there is no problem because all of them work in a sufficiently high frequency. However, if for example a landmark detector that has long times for a detection is implemented, a time synchronization would be needed to relate its detection to the corresponding node.
- *No loop closure constraint validation*: It is widely known that a bad loop closure constraint can severally affect the quality of a map. However the current version of the framework does not check this condition and the responsibility is on the user.
- *No memory management system*: The memory management system was left out of the scope of the project because it is possible to obtain a modular SLAM framework without it. Nonetheless, for the implementation on a real robot it is highly recommended to allow the creation of larger maps.

## 6.3 Recommendations for future work

The goals of this project were to identify the essential functionalities of a modular SLAM system and design a framework that could be implemented in state of the art solutions like RTAB-Map. These goals were achieved, but the framework is in its first version and could be further improved to include more complex functionalities present in other non-modular solutions. The recommendations in this section can be divided in two:

1. The main recommendation is to complete the integration with RTAB-Map. The framework proved to be compatible with RTAB-Map, but in order to be considered by other researchers the integration must be complete. This task will also require further testing and debugging before it can be shared with the community, i.e on Github.

2. The next points to be considered for future work are the already mentioned features that are commonly seen in other SLAM frameworks:

    - Study the inclusion of metric/sensor data to the map: Most SLAM frameworks include this kind of data into the map. This allows the creation of occupancy grids, and other representations of the map which are used in many SLAM applications. However, a deeper study should be carried out in order to asses the modularity concerns of this implementation.
    - Addition of a memory management system: RTAB-Map uses a memory management system that allows the framework to work on real-time constraints in large maps. However, as seen in Section 3.2, this system creates a lot of unnecessary dependencies due to the usage of visual data. For this reason, in our framework, a memory management system using a different criteria not dependent on the front-ends could be implemented. For instance, using the timestamp or proximity data of the nodes.
    - Constraint validation and input data synchronization: As stated in the discussion, the framework would benefit from the implementation of a constraint validation and an input data synchronization functions.

# A Appendix: Common Front-ends in SLAM

There are many kind of sensors used to solve the SLAM problem, but among the most popular are the LiDAR and the camera. The data that these sensors provide can be processed in many different ways to be used in SLAM, in this section it will be explained the techniques that are most commonly used in the state of the art.

## A.1 Visual odometry

Visual odometry is a particular case of the problem structure from motion SFM, widely known in the computer vision community. SFM consists in recovering relative camera poses and the 3D structure from a set of camera images, which its global optimization is usually performed offline due to the high computational expenses. Whereas in VO, the 3D motion of the camera is recovered sequentially as new images arrive and local optimization can be performed online.

### A.1.1 Feature-based methods

A classical solution to VO is the one explained in Scaramuzza and Fraundorfer (2011). This solution consists in extracting a set of features $f$ (either points or lines) from every image $I$ and estimating the transformation $T$ between the current time instant $k$ and the previous time instant $k-1$. Depending on the number of dimensions in which the feature correspondences are specified, there are different methods:

- 2D-to-2D: In this case, the features $f_{k-1}$ and $f_k$ are specified in 2D image coordinates. To obtain the transform $T_k$ the Essential matrix is computed using the epipolar constraint. This matrix can be acquired using the 8-point algorithm or the 5-point algorithm, where 8 and 5 feature correspondences are found respectively. The correspondences can be found using different **feature matching algorithms** and the random sample consensus (RANSAC) is often used for outlier rejection. The essential matrix is decomposed into the rotation matrix $R$ and the translation vector $t$. The relative scale is then obtained from two consecutive transformation to approximate the absolute scale of the translations. The information is concatenated to recover the trajectory.

- 3D-to-3D: In this case, the feature are specified in 3D image coordinates $X_{k-1}$ and $X_k$. It is necessary to triangulate the 3D points previously to the transform, for example using a RGB-D or stereo camera. In this case a minimum of three noncollinear correspondences are needed, and the transformation is the one that better aligns the two sets of 3D points. This transformation can be achieved by minimizing the feature position error (A.1) using a least-squares approximation. The transformation is in absolute scale, so the trajectory can directly composed concatenating the transforms.

$$\arg\min_{T_k} \sum_i \left\| \tilde{X}_k^i - T_k \tilde{X}_{k-1}^i \right\| \tag{A.1}$$

- 3D-to-2D: In this case, we look for the transform that minimizes the reprojection error between the 3D feature points $X_{k-1}$ from the previous instant and the current 2D features $f_k$ in $I_k$. This problem is called perspective from n points (PnP), and a common solution is the P3P where three correspondences are used. In the monocular case, three images are needed, two for the triangulation of $X_{k-1}$ and a third one to find the projections in it. This method gets a more accurate estimation than the 3D-to-3D case, due to the minimization of the reprojection error (A.2).

$$\arg\min_{T_k} \sum_i \left\| p_k^i - \hat{p}_{k-1}^i \right\|^2 \tag{A.2}$$

### A.1.2 Featureless/ Direct methods

There are two kinds of featureless methods: dense (if they exploit the information in every pixel) or semi-dense (if they only use the pixels in which the gradient of the image's brightness is significant).

In FAIA (Forward Additional Image Alignment) the squared pixel intensity difference.

$$\underset{p}{\operatorname{argmin}} \sum_{x,y} [I(W(x,y,p)) - J(x,y)]^2 \tag{A.3}$$

There are others like FCIA (forward Composition Image Alignment), ICIA (Inverse Composition Image Alignment) and IAIA (Inverse Additional Image Alignment), which are presented in the work of Baker and Matthews (2004).

### A.1.3 Visual loop-closure

Most of the visual SLAM algorithms solve the data association problem differently for visual odometry and for loop closure detections. In the last case the loop closure is assisted by place recognition techniques. The most common approach is the use of a BoVW (like the one used inLabbe and Michaud (2013)). Once a loop closure hypothesis is generated, it can be validated or directly used for a new constraint generation using the methodologies explained in A.1 depending on the available data.

## A.2 LiDAR odometry

Similarly to visual odometry, LiDAR odometry solves the problem of reconstructing the trajectory of the sensor platform by iteratively computing the relative transformation between sequences of LiDAR clouds $P$ in consecutive time instants $k-1$ and $k$. Note that now, because of the nature of the LiDAR sensor, the input data represents points in 3D space, either all in the same plane in the case of 2D Lidars or in the whole space in the case of 3D Lidars. Most of the LiDAR odometry approaches are based in the work of Lu and Milios (1997).

### A.2.1 Feature-based methods

Firstly, as it happened in visual odometry, there is a feature-based approach. LiDAR odometry can also extract features from the input data and solve the data association problem with known correspondences. Although the methods for feature extraction and data association are different because now we do not have image information. A popular implementation of a feature-based scan-matching algorithm is the one in Zhang and Singh (2014), where the extracted features are denoted by sharp edges and planar surface patches. Once a feature correspondence is found, the distance between the feature and the correspondence is computed for every pair. The motion is estimated by minimizing the overall distances. Moreover, robust pose estimation can also be achieved in LiDAR odometry, for instance rejecting outliers by adding weights to the feature correspondences depending on the distance. This procedure is the same that the one described in the 3D-to-3D case in visual odometry.

### A.2.2 Featureless/ Direct methods

Another possible solution is performed without the data association step, also called a featureless scan-matching, but this time no optimal nor direct solution exists. The approach in this case is based on the Iterative Closest Point (ICP) algorithm, which consists in guessing some point correspondences and try to minimize the distance between them multiple times until the error is minimized under a threshold. Note that this approach depends in the assumption of having an initial guess. This initial guess can be provided by other sources of odometry or by some knowledge of the potential correspondences. This methodology have some disad-

vantages, which are mainly caused by the iterative nature of the algorithm and the need of a guess. The result can be very degraded if there are bad correspondences and also many iterations might be requires. Nevertheless, several authors worked on the speed and robustness of the algorithm and different variants are commonly used in the state of the art (i.e Kohlbrecher et al. (2011) or Hess et al. (2016)).

$$\sum_{i=1}^{n} \| M_i - (RS_i + t) \|_2^2 \tag{A.4}$$

### A.2.3 LiDAR loop closure

Featureless and feature-based algorithms can be used to obtain a loop-closure in LiDAR-based SLAM. Nevertheless, comparing the current scan with all the previous data would require a lot of computations and it is not feasible for real-time operation. Most of the current state of the art solves this problem by taking into account the stored topological information and use a sliding window to only take into account for a loop closure the places that are near the robot. Other loop closure hypothesis generation methods exist, like the one proposed by Magnusson et al. (2009). This study proposes a Normal Distribution Transform (NDT) representation of the 3D pointclouds to create feature histograms that represent the surface orientation and smoothness. This histograms can then be matched to detect loop closures, similar to the BOVW approach in the visual loop closure detection.

# Bibliography

Agarwal and Mierle (2010), Ceres Solver, http://ceres-solver.org.

april tags (2019), April tag 3D models, https://github.com/koide3/gazebo_apriltag.

Aziobot (2021), Aziobot Sb2 robot., https://aziobot.com/.

Baker, S. and I. Matthews (2004), Lucas-kanade 20 years on: A unifying framework, in *International journal of computer vision*, volume 56, Springer, pp. 221–255.

Blanco-Claraco, J.-L. (2019), A Modular Optimization Framework for Localization and Mapping., in *Robotics: Science and Systems*.

Boston dynamics (2021), Boston dynamics' Spot robot., https://www.bostondynamics.com/spot.

Brommer, C., D. Malyuta, D. Hentzen and R. Brockers (2018), Long-Duration Autonomy for Small Rotorcraft UAS Including Recharging, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, p. arXiv:1810.05683, doi:10.1109/iros.2018.8594111. https://doi.org/10.1109/iros.2018.8594111

Cadena, C., L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid and J. J. Leonard (2016), Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age, in *IEEE Transactions on robotics*, volume 32, IEEE, pp. 1309–1332.

Colosi, M., I. Aloise, T. Guadagnino, D. Schlegel, B. Della Corte, K. O. Arras and G. Grisetti (2020), Plug-and-Play SLAM: A Unified SLAM Architecture for Modularity and Ease of Use, in *arXiv preprint arXiv:2003.00754*.

Colosi, M., S. Haug, P. Biber, K. O. Arras and G. Grisetti (2019), Better Lost in Transition Than Lost in Space: SLAM State Machine, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 362–369.

Dellaert, F. (2012), Factor graphs and GTSAM: A hands-on introduction, in *GT-RIM-CP&R-2012-002*, Georgia Institute of Technology.

Dennis Ellery (2017), "Writing reusable code for robotics", master's thesis, University of Twente.

gaz (2020), Gazebo 11, http://gazebosim.org/.

Grisetti, G., R. Kümmerle, C. Stachniss and W. Burgard (2010), A tutorial on graph-based SLAM, in *IEEE Intelligent Transportation Systems Magazine*, volume 2, IEEE, pp. 31–43.

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

Hess, W., D. Kohler, H. Rapp and D. Andor (2016), Real-time loop closure in 2D LIDAR SLAM, in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 1271–1278.

Jeroen Minnema (2020), "Towards component-based, sensor independent and back-end independent SLAM", master's thesis, University of Twente.

Joost van Smoorenburg (2020), "Creating a RTAB-Map variation for implementation in a component-based SLAM framework", individual assignment, University of Twente.

Kohlbrecher, S., O. Von Stryk, J. Meyer and U. Klingauf (2011), A flexible and scalable SLAM system with full 3D motion estimation, in *2011 IEEE international symposium on safety, security, and rescue robotics*, IEEE, pp. 155–160.

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

Labbé, M. and F. Michaud (2011), Memory management for real-time appearance-based loop closure detection, in *2011 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, pp. 1271–1276.

Labbe, M. and F. Michaud (2013), Appearance-based loop closure detection for online large-scale and long-term operation, in *IEEE Transactions on Robotics*, volume 29, IEEE, pp. 734–745.

Labbe, M. and F. Michaud (2014), Online global loop closure detection for large-scale multi-session graph-based SLAM, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 2661–2666.

Labbé, M. and F. Michaud (2018), Long-term online multi-session graph-based SPLAM with memory management, in *Autonomous Robots*, volume 42, Springer, pp. 1133–1150.

Labbé, M. and F. Michaud (2019), RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation, in *Journal of Field Robotics*, volume 36, Wiley Online Library, pp. 416–446.

Laurent Gomila (2021), SFML 2.5, https://www.sfml-dev.org.

Lu, F. and E. Milios (1997), Robot pose estimation in unknown environments by matching 2d range scans, in *Journal of Intelligent and Robotic systems*, volume 18, Springer, pp. 249–275.

Lynen, S., M. W. Achtelik, S. Weiss, M. Chli and R. Siegwart (2013), A robust and modular multi-sensor fusion approach applied to mav navigation, in *2013 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, pp. 3923–3929.

Magnusson, M., H. Andreasson, A. Nuchter and A. J. Lilienthal (2009), Appearance-based loop detection from 3D laser data using the normal distributions transform, in *2009 IEEE International Conference on Robotics and Automation*, IEEE, pp. 23–28.

Malyuta, D., C. Brommer, D. Hentzen, T. Stastny, R. Siegwart and R. Brockers (2019), Long-duration fully autonomous operation of rotorcraft unmanned aerial systems for remote-sensing data acquisition, *Journal of Field Robotics*, p. arXiv:1908.06381, doi:10.1002/rob.21898.
https://doi.org/10.1002/rob.21898

Mark te Brake (2021), "Modular SLAM improvements for the Real-Time Appearance Based Mapping (RTAB-Map) algorithm", master's thesis, University of Twente.

Mathieu Nass (2020), "Wi-Fi based range-only constraint integration in RTAB-Map algorithm", bachelor's thesis, University of Twente.

Mohamed A. Abdelhady (2017), "Reuse-oriented SLAM Framework using Component-based Approach", master's thesis, University of Twente.

ROBOTIS (2021), Turtlebot3, https://github.com/ROBOTIS-GIT/turtlebot3.

ROS (2021), ROS, https://www.ros.org/.

Scaramuzza, D. and F. Fraundorfer (2011), Visual odometry [tutorial], in *IEEE robotics & automation magazine*, volume 18, IEEE, pp. 80–92.

SFPlot (2021), SFPlot, https://github.com/MoriokaReimen/SFPlot.

Sturm, J., N. Engelhard, F. Endres, W. Burgard and D. Cremers (2012), A benchmark for the evaluation of RGB-D SLAM systems, in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, pp. 573–580.

Tesla (2021), Tesla model Y., https://www.tesla.com/es_es/modely.

Tim Broenink (2016), "On reusable SLAM", master's thesis, University of Twente.

Wang, J. and E. Olson (2016), AprilTag 2: Efficient and robust fiducial detection, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 4193–4198, ISBN 978-1-5090-3762-9, doi:10.1109/IROS.2016.7759617.

Zhang, J. and S. Singh (2014), LOAM: Lidar Odometry and Mapping in Real-time., in *Robotics: Science and Systems*, volume 2.