**POLITECNICO DI MILANO**
**MSc in Computer Science and Engineering**
**Scuola di Ingegneria Industriale e dell'Informazione**
**Dipartimento di Elettronica, Informazione e Bioingegneria**

# AN EXPERIMENT IN AUTONOMOUS NAVIGATION FOR A SECURITY ROBOT

**AI & R Lab**
**Laboratorio di Intelligenza Artificiale**
**e Robotica del Politecnico di Milano**

**Supervisor: Prof. Matteo Matteucci**
**Co-supervisor: Ing. Gianluca Bardaro**

**Master Graduation Thesis by:**
**Fabio Santi Venuto,**
**Student ID 837644**

**Academic Year 2016-2017**

*Alla mia famiglia...*

# Abstract

One of the most useful purpose in autonomous robots is to substitute for human activity in the so called Dirty, Dangerous, Dull (DDD) tasks. The Ra.Ro platform, developed by NuZoo, is designed to work as a security robot, able to patrol, detect anomalies, and eventually send alarms to a centralized station. This platform is has to be adapted to different purposes and customized for each client, but in most of cases an autonomous navigation is required. The aim of this thesis is to propose a mapping and localization system to avoid the well-known odometry drifting problem and allow for long term autonomous navigation the Ra.Ro. platform. In particular, the robot moves in an indoor environment, challenging because of the lack of any global positioning sensor such as GPS in outdoor. The odometry system provided by the wheel encoders is not precise enough and very sensitive to errors, thus it is important to fuse the information retrieved by multiple sensors such as IMU, LIDAR and a camera used to recognize specific markers in order to compensate for odometry drifting.

The final results, testing the robot in real world scenarios, are quite satisfying, allowing finally the robot to move autonomously in an environment previously mapped.

# Sommario

I robot autonomi nel ruolo di guardia di sicurezza non sono ancora comuni. Una guardia efficiente deve essere abile nell'individuare persone non autorizzate o qualsiasi altra cosa che non vada bene. Deve essere reattiva, veloce e ovviamente difficile da battere. Probabilmente la tecnologia attuale è prematura, ma la piattaforma Ra.Ro. è molto semplice da adattare a scenari differenti, a seconda delle richieste dei clienti. Essendo un robot basato su una meccanica *Skid-steering*, ha senza dubbio l'abilità di muoversi autonomamente nell'ambiente, a prescindere dalla sua finalità.

La navigazione "autonoma" del Ra.Ro. commercializzato finora consiste nel seguire delle linee colorate incollate o dipinte sul pavimento e/o seguire delle indicazioni date da dei marker appartenenti ad uno specifico set e riconosciuti dalle telecamere installate. I problemi di questi approcci sono faclimente individuabili. Prima di tutto, in determinati luoghi, non è desiderabile avere il pavimento rovinato da linee incollate o dipinte, mentre all'aperto è praticamente impossibile disegnarle o incollarle. Inoltre diverse condizioni luminose possono condizionare il riconoscimento del colore delle linee. Possiamo avere problemi simili quando abbiamo a che fare con i marker, che hanno bisogno di essere appesi su muri o altri tipi di strutture stabili. Se il robot usa le linee e i marker per pattugliare un edificio per ragioni di sicurezza, sarebbe facile per un malintenzionato coprire, cancellare o staccarli, facendo perdere il robot in pochi secondi.

# Ringraziamenti

Poche parole, anche perché di più non me ne verrebbero. Sicuramente i miei genitori sono le persone a cui vanno i miei più sentiti ringraziamenti. Sarà banale dirlo, ma senza di loro e il loro supporto non sarei qui. Mio padre dice sempre che "La libertà è avere le possibilità" e loro sono stati in grado di darmi la libertà necessaria e sono sicuro che continueranno a farlo. Grazie a mia sorella, Valeria, che mi sa capire e che nel momento del bisogno c'è sempre.

Un ringraziamento speciale alla mia ragazza, Desy, che mi ha accompagnato durante questo lungo percorso, dal primo giorno di università, fino all'ultimo, aggiungendo un pizzico di amore che ha reso la strada da percorrere più piacevole.

Grazie agli amici di una vita, i miei amici pievesi (ed ex pievesi); Ori, Andre, Mane e Gabry, e i nuotatori in pensione; Paolo, Giò, Andreino e Teo. Di cuore a tutti un abbraccio. Insostituibili.

Grazie a mio cugino, Alessandro, che da quando è tornato in Italia mi ha riavvicinato ad un pezzo di cultura: i videogiochi :) Forse ora avremo più tempo per giocare a Overwatch con Alberto.

Grazie agli amici incontrati in università, in particolare ai ragazzi dell'AirLab, Ewerton, Enrico, Davide e Dave, Teo e Luca e tutti coloro che sono passati da lì. Hanno reso l'ambiente di lavoro davvero speciale e il loro supporto morale e tecnico è stato fondamentale.

Infine, ma non ultimi, grazie ai professori, in particolare il mio relatore, Matteucci, una persona che mille ne pensa e duemila ne fa -letteralmente- e Gianluca che mi hanno fatto capire un po' di più cosa vuol dire essere un ingegnere informatico.

Grazie,

Fabio

# Contents

# List of Figures

# Chapter 1

# Introduction

*"Narrator: Deep in the Caribbean, Scabb Island.*
*Guybrush: ...So I bust into the church and say, "Now you're in for it, you bilious bag of barnacle bait!"... and then LeChuck cries, "Guybruysh! Have mercy! I can't take it anymore!"*
*Fink: I think how he must have felt.*
*Bart: Yeah, if I hear this story one more time, I'm gonna be crying myself."*

Monkey Island 2: LeChuck's revenge

Autonomous robots as security guardian are not yet common. An efficient guard must be very smart in detecting unauthorized people or anything else wrong. It must be reactive, fast and of course hard to be defeated. Probably the current technology is premature to perform this task, but the Ra.Ro. platform is very easy to be adapted to different scenarios, according to the customers' requests. Being a skid-steering based robot, it has certainly the ability to move autonomously in the environment, regardless of its high-level purpose.

Until now, the "autonomous" navigation of the commercialized Ra.Ro.s consists in following colored lines sticked or painted on the floor and/or following indications given by markers belonging to a specific set and recognized by the installed cameras. The issues of these approaches are easy to identify. First of all, in some locations it is not desirable to have the floor ruined by sticked or painted lines while in outdoor environments it is almost impossible to draw or stick them. Moreover, different light conditions can affect the line color detection. We can have a similar issue when dealing with markers, which need to be hanged on walls or on similar stable structures. If the robot uses the lines and markers to patrol a building for security reasons, it will be quite easy for an malitious person to cover, delete or detach them, getting the robot lost in seconds.

## 1.1 Thesis contribution

The Ra.Ro. platform is already equipped with different sensors such as IMU (gyroscope, accelerometer and magnetometer), cameras and a LIDAR, but they were not used as much as they could be potentially done, especially these are not used for navigation.

This thesis proposes a multi-sensor navigation system based on the ROAMFREE framework, a system that provides a multi-sensor fusion tools to improve the odometry estimation using the information provided by different sensors. We used the wheel encoders, gyroscope, accelerometer and, for a better result, visual markers as landmarks. Then, using the improved odometries, we generated and compared different maps using mostly the `gmapping` framework. Subsequently we set up the autonomous navigation system using the `move_base` framework with AMCL and we tested the whole navigation stack in our indoor environments.

## 1.2 Structure of the thesis

The thesis is structured as follows:

- In Chapter 2 we describe the Ra.Ro. platform. In particular in Section 2.1 we introduce the hardware components which include the sensors provided with the robot. In Section 2.2 we introduce the robot built in software environment. In particular we describe the ROS topics used by the robot and the built in navigation systems in the following subsections.

- In Chapter 3 we provide the background knowledge needed to deeply understand in what our project consists. In particular, Section 3.1 introduces the most common state estimation approaches. Then, in Section 3.2, we focus on odometry estimation methods. Subsequently, in Section 3.3 we introduce the sensor fusion approaches we applied in this thesis, in particular we will introduce a custom odometry generation method and the ROAMFREE framework. The following Section 3.4 is about SLAM, simultaneous localization and mapping, and two of the most popular ROS mapping systems. In Section 3.5 we describe the AMCL localization algorithm. We conclude with a note about the ROS reference system convention, in Section 3.6

- In Chapter 4 we describe our contribution, starting with the navigation system overview in Section 4.1. Then, in Section 4.2 we describe the

detail about the odometry estimation implementation and the sensor fusion framework applied. In Section 4.3 we explain how we used the mapping module.

- In Chapter 5, after a brief set up introduction in Section 5.1, we describe the most significant experiment we make and the resulting output. In particular, we describe experiments about odometry estimation, in section 5.2, about mapping, in Section 5.3, and we provide a brief discussion about localization and autonomous navigation in section 5.4

- Finally in Chapter 6 we write about our conclusion, the main challenges we had to deal with, and our suggestion about future works on this topic.

# Chapter 2

# The Ra.Ro. platform

*"Guybrush: My name's Guybrush Threepwood. I'm new in town*
*Pirate: Guybrush Threepwood? That's the stupidest name I've ever heard!*
*Guybrush: Well, what's YOUR name?*
*Pirate: My name is Mancomb Seepgood."*

The Secret of Monkey Island

In this chapter we introduce the Ra.Ro. platform. Ra.Ro. stands for Ranger Robot, that is to say that its main purpose is to work as a security guard, able to patrol parkings, supermarkets and so on. However, the producer company, NuZoo offers the possibility of customizing the platform to meet different requests. It has already been adapted by the producer company for other different purposes, usually starting from its simpler version, code name Geko, which is very similar to Ra.Ro. except for the cameras, which are attached directly to the robot basis.

In the following paragraphs we focus on the version we have worked on, introducing its hardware and software suite. [1]

## 2.1 Ra. Ro. Hardware

Ra.Ro. is a skid steer drive robot. The basis is 460 mm x 540 mm and it is 270 mm tall. The robot reaches 750 mm including the cameras support. It is equipped with four wheels driven by two 50W DC motors, set in the middle of the two sides of the robot. Each motor drives a front and a rear wheel connected with two transmission belts.

The robot is equipped with a 9 axis IMU composed by a LSM303D module (3 axis magnetometer and 3 axis accelerometer) and a L3GD20H (3 axis gyroscope). The IMU is managed by a R2P board by Nova Labs [2]. A

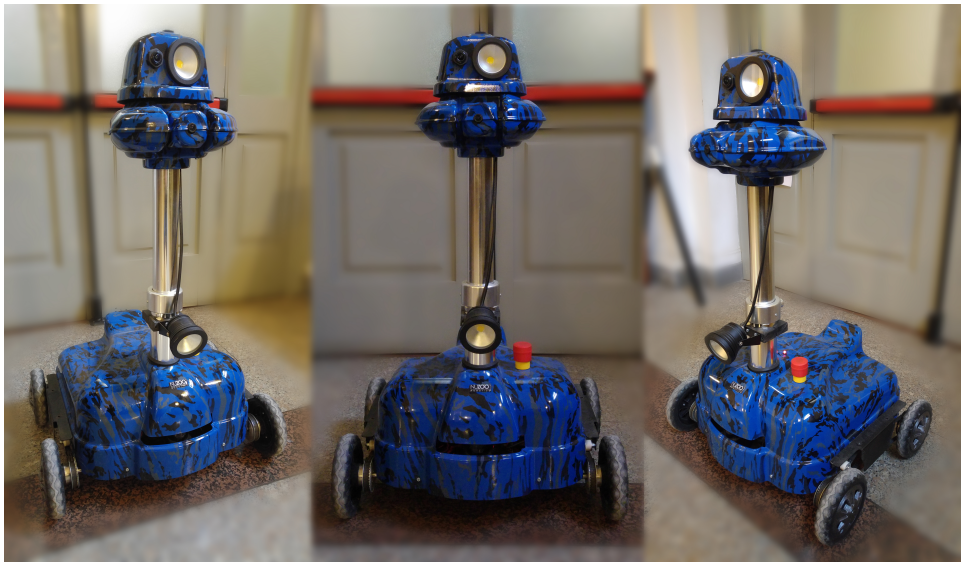


*Figure 2.1: Three Ra.Ro. views*

Hokuyo Laser Scanner is included inside the robot body and allows a 170° view by projecting rays trough a 160 mm wide and 20 mm high body slit.

Inside the "head" of the Ra.Ro. are two HD cameras: a surveillance camera, which rotates according to the head itself and a navigation camera, which can, in addition, rotate around a horizontal axis. Vision into the darkness is guaranteed by two led flashlights. We can see in Figure 2.1 three Ra.Ro. views in which is possible to notice the two cameras, one flashlight inside the head, one near the basis and the body slit at the bottom, from which infrared laser rays are projected.

The core of the robot is an INTEL NUC built with an i5-5250U 64bit CPU, DDR3 4 GB RAM and SSD 60GB as a hard drive support. A Wi-Fi module is used to connect the robot to wireless nets or to convert it into an access point, in case of accessible net unavailability or first net setup. Ra.Ro. is able to recharge itself in a semi-autonomous way through a wide contacts-pins matching with its recharging station. In case of needs, a wired connection is also available to recharge the robot.

## 2.2 Ra.Ro. Software

The operative system currently installed on the NUC is Ubuntu 14.04 LTS, and all the robot features are managed by ROS Indigo. The provided ROS workspace includes most of the nodes and topics needed to start our work. In particular, we have nodes responsible for publishing sensors data, such as

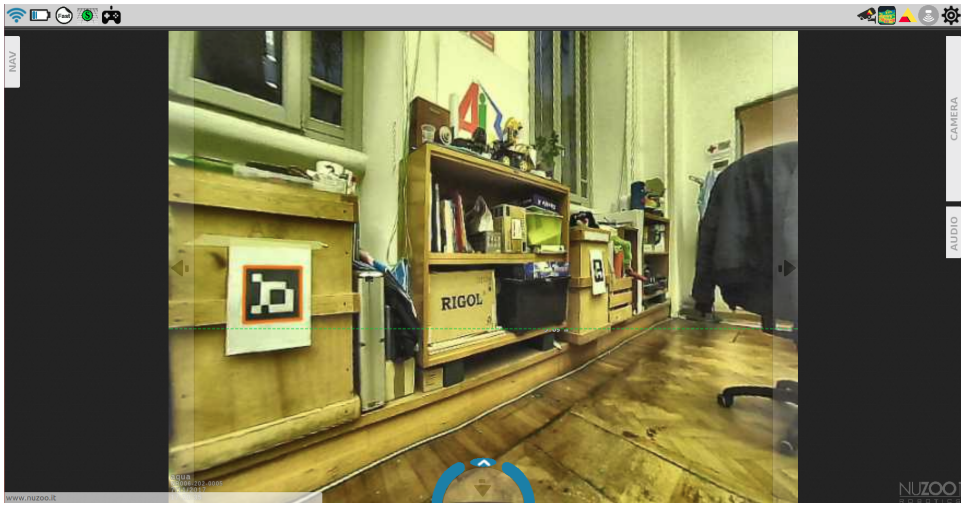



*Figure 2.2: NuZoo web interface. The recognized marker is orange bordered.*

encoders, IMU, camera vision, laser scans and markers. Moreover, we have the nodes used to control the robot through the command velocity topics, via joy-pad or via browser interface. The browser interface itself is also a useful piece of software, which allows the user to see the images provided by the two cameras, the recognized marker, and to manage the various minor functionalities such as speakers, lights and so on. In Figure 2.2 we can see the camera view form the browser interface. When a marker is recognized, like in this case, an orange border is overlapped around the marker, in the camera view.

### 2.2.1 ROS topics

In this section we describe the most relevant, to our purposes, ROS topics published and/or subscribed by the previously implemented nodes. They allow the ROS system to communicate both with sensors and actuators, by reading outputs and sending commands respectively. The topics are introduced with the name and the message type used.

- `/r2p/encoder_l` and `/r2p/encoder_r`, `[std_msgs/Float32]`:
  These topics, and all of the following ones which have a name starting with `r2p/`, are published by the r2p board, which manages most of the sensors. In particular, these topics publish as messages the number of ticks per second done by the left wheels (`r2p/encoder_l`) and the right ones (`r2p/encoder_r`). Every tick corresponds to a portion of spun wheel.

- `/r2p/imu_raw`, `[r2p_msgs/ImuRaw]`:
  This topic contains the raw messages from the IMU, including gyroscope, accelerometer and magnetometer. The `r2p_msgs/ImuRaw` is a custom message type built by three tridimensional vectors: `angular_velocity`, `linear_acceleration` and `magnetic_field`. The names are self-explanatory enough and every vector has one component per axis: `x`, `y`, `z`. The values published represent the MEMS sensor register copy. In particular, the gyroscope has a 16 bit reading, covering a range from -500 dps to +500 degrees per second (dps), with a sensitivity of 17.50 millidegree per second (mdps) per least significant bit (LBS) and the accelerometer has a 12 bit reading, covering the range from $-2g$ to $+2g$, that is about $1mg$ per LSB. The $g$ here stands for gravitational acceleration which measures about 9.81 $m/s^2$

- `/r2p/odom`, `[geometry_msgs/Vector3]`:
  The r2p board system publishes in this topic a very raw odometry. It is retrieved only by encoders data elaborations and is published as a vector of three elements in which the first and the second elements represent the position variation in meters (`x` and `y`) and the third element represents the orientation angle variation in radians.

- `/nav_cam/markers`, `[nav_cam/MsgMarkers]`:
  The `nav_cam` node publish in this topic the list of markers recognized by the robot, in real time. Each marker is represented as a `nav_cam/MsgMarker` message, which includes a numerical id of the marker (`id`), the name of the published marker frame (`frame_id`) and its transform with the respect of the camera frame (`pose`), divided in `position`, as a tridimensional vector, and `orientation`, as a quaternion.

- `/odom`, `[nav_msgs/Odometry]`:
  The `autonomous_nav` node publishes the odometry built using the messages from `/r2p/odom`. We have improved this odometry for this project, as we will explain in Chapter 4. The `nav_msgs/Odometry` message contains `pose` information, divided in `position` and `orientation`, and `twist` information, i.e. velocity, which is divided in `linear` and `angular`, both with the respective `covariance` matrices.

- `/scanf`, `[sensor_msgs/LaserScan]`:
  The `autonomous_nav` node also publishes messages from the Hokuyo laser scanner, after being filtered by some outliers. The

*Figure 2.3: Ra.Ro. photograph from NuZoo website*

`sensor_msgs/LaserScan` messages represent the collection of the distances at which an infrared ray beamed by the laser scanner is intercepted by an obstacle.

### 2.2.2 Built-in navigation

The provided workspace includes different ways to teleoperate the robot. All of them always use the so called "laser bumper" which basically interrupts all forward movements in case of obstacle detected by laser scanner in a very close range. The teleoperation system operates in three ways: manual, assisted and semi-autonomous.

**Manual teleoperation** : The manual teleoperation is managed through the web application using a remote controller being plugged into a computer and connected to the robot via network. It is the simplest way and it relies completely on human control.

**Assisted teleoperation** : The assisted teleoperation is managed via the web application interface which in this case can be run even on mobile devices. It is Google street view inspired and allows to move the robot towards a specific spot by clicking or touching on the correspondent on-screen spot in the map besides its basic movements such as forward, backward and left and right rotation. The system cannot manage obstacles in the trajectory. We can also consider as assisted teleoperation the one with the particular *follow me* marker. The operator can show the marker to the robot's navigation camera and the robot tries to keep the marker into the camera frame, following the person is holding it.

**Semi-autonomous teleoperation** : The most advanced navigation systems built into the robot consist in line following and in marker indication following. The first one consists in following colored line sticked or painted on the floor. It is possible to switch from one color to another. The second one consists in executing simple navigation tasks according to the recognition of a specific marker, which can be attached in sequence on walls creating, if needed, a patrol path. Examples of a marker command can be to turn left 90°, turn 180°, keep the wall on your right. Moreover, a special marker is set on the recharging station and the robot can, under proper conditions, autonomously connect itself to the station after recognizing the marker.

None of these system implements a proper obstacle avoidance algorithm, nor a good odometry calculation. There is no mapping system, thus no localization is possible.

# Chapter 3

# Background knowledge

*"Smirk : I like your spirit. I'll do what I can. Of course... it'll cost you. What have you got?*
*Guybrush : All I have is this dead chicken.*
*Smirk : That isn't one of those rubber chickens with a pulley in the middle is it? I've already got one. What ELSE have you got?*
*Guybrush : I've got 30 pieces of eight.*
*Smirk : Say no more, say no more. Let's see your sword.*
*Guybrush : I do have this deadly-looking chicken.*
*Smirk : Yes, swinging a rubber chicken with big metal pulley in it can be quite dangerous... BUT IT'S NOT A SWORD!!! Let's see your sword."*

The Secret of Monkey Island

## 3.1 State estimation

For a robot interacting with the environment it is important to retrieve information about the state of the environment around it and its own state. This knowledge cannot be summarized into a unique hypothesis, in fact it is crucial to also have a characterization of the uncertainty of this knowledge [3].

We define the *state* of a robot as the values of specific variables needed to identify the robot and/or parts of it in a specific state space, for example, velocity, position or orientation of a particular component. Most of the contemporary autonomous robots represent the possible states using probability distributions in order to not rely only on a single "best guess", but to have the possibility to take decisions under uncertain conditions.

A well designed robot should be able to retrieve heterogeneous information given by different kind of sensors, in order to make the update of the possible states depending on complementary information.

This is basically the definition of *sensor fusion* and in the following paragraph we are going to introduce the most used probabilistic techniques for state estimation. These techniques are employed in different fields but on this essay we will focus only on the robotic field.

### 3.1.1 Baysian state estimation

We define the *belief* of a state with the following formula:

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

the belief of a state at time $t$ is defined as the probability to be in that state given the measurements $z$ from the sensors and the known input values $u$ until the time $t$.

To update the state and retrieve the measurement of the sensor at the $t$ time is not very practical; Indeed, the formula

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t})$$

is more often used, in wich a *posterior* distribution represents the probability of each state given its *prior*, i.e. sensor readings and the controls until time $t-1$. Hence, this distribution is called *prediction*. From $\overline{bel}(x_t)$ we can obtain $bel(x_t)$ in a recursive fashion, considering the first term as prior and the second one as posterior. The most general form of the recursive state estimation is the Bayes filter, which is here reported in Algorithm 1.

For each state variable we can divide its estimation into two steps. In the first one, *prediction*, we estimate the state variable at time $t$ given the $u_t$ and $x_{t-1}$. In the second one, *update*, the sensor readings $z_t$ are used, combined with the previously calculated prediction. The mathematical formulation of the Bayes filter is given by

$$bel(x_t) = \eta p(z_t|x_t) \int_x p(x_t|x_{t-1}, u_t) p(x_t) dx_{t-1}$$

which can be obtained:

- from the Bayes rule: $\frac{P(B|A)P(A)}{P(B)}$, together with the law of total probability:
$$P(A) = \int P(A|B)P(B)dB$$
- assuming that the states follow a first-order Markow process, i.e. past and future data are independent if the current state is known:
$$p(x_t|x_{0:t-1}) = p(x_t|x_{n-1})$$

**Algorithm 1** Bayesian filtering

1: **function** BAYESFILTER($bel(x_{t-1}), u_t, z_t$)
2: **for all** $x$ **do**
3: $\overline{bel}(x_t) \leftarrow \int_x p(x_t|x_{t-1}, u_t)p(x_t)dx$
4: $bel(x_t) \leftarrow \eta p(z_t|x_t)\overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$
7: **end function**

- assuming that the observations are independent of the given states, i.e.

$$p(z_t|x_{0:t}, z_{1:t}.u_{1:t}) = p(z_t|x_t)$$

The generic Bayes filter algorithm is almost impossible to use since the analytical representation of the multivariate posterior is usually difficult to compute. Moreover, the integrals involved in the prediction represent a very high computational effort.

The first practical implementation of the Bayesian filter for continuous domains was made by Rudolph E. Kalman, in 1960. The original formulation assumes that the belief distribution and the measurements noise follow a Gaussian distribution and that system and observation models are linear [4]. Under these assumptions, the Kalman update equation yields the optimal state estimator, in terms of mean squared-error. The so called Kalman Filter (KF) is very important and it is still considered the state of the art in state estimation, especially its more generic version, the Extended Kalman Filters (EKFs), which admit the non-linearity of the system. The EKF solution lies in a Taylor series expansion applied to linearize the requested functions.

These solutions i.e. KF and EKF, are still widely employed today and are often the first choice in recursive state estimation. However, none of them KFs holds in the non-linear case. In particular, the EKFs can suffer from a poor approximation caused by the linearization of highly non-linear models affected by the propagation of the Gaussian noise. In 1997, Simon Julier and Jeffery Uhlmann proposed the Unscented Kalman Filter (UKF) [5] which, using the *unscented transform* for the linearization, can obtain better results in terms of accuracy, keeping the error characterization as a Gaussian noise, which is usually reliable enough, and above all, easy to represent since the mean and the covariance give the full description of the distribution.

An alternative to the Kalman Filters is given by non-parametric filters. These do not rely on an analytic representation of the posterior probability distribution, nor on parametric distributions. A well-known non parametric

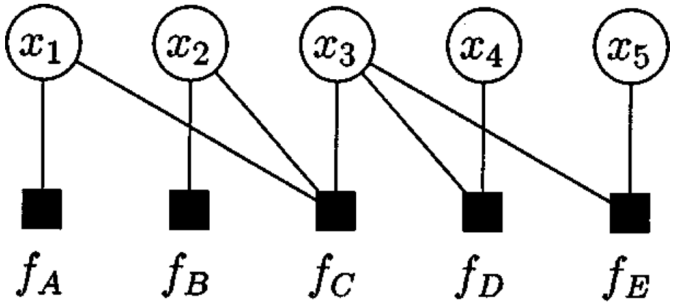



*Figure 3.1: A factor-graph for the product* $f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3)f_D(x_3, x_4)f_E(x_3, x_5)$.

approach is the *particle filter*, proposed by Gordon et Al. in 1993 [6]. The idea is to describe the posterior distribution in a Monte Carlo fashion, representing a possible state with a *particle*. The more particles are present in a certain region of the state space, the more that state is likely to be the real state. The advantage of this approach is that any kind of distribution can be represented in this way, but there are still some issues. In particular, we have to deal with a possible high dimension of the state space which carries the exponential growth of the number of particles needed to represent the probability distribution of the belief.

### 3.1.2 Graph-based State Estimation

In 1997, Lu and Milios developed and proposed a graph-based approach [7] to solve the Simultaneous Localization and Mapping (SLAM) problem. We discuss about SLAM in Section 3.4. Here we want to focus about the proposed graph based approach. In their formulation, the nodes in the graph represent poses and landmark parameterizations. If a landmark is visible from a certain pose, then an edge linking the two poses is added. The state estimation problem consists in a global maxi-likelihood optimization. Since every node and edge represents respectively poses and landmarks, the aim is to maximize the observations joint likelihood. This requires to solve a large non-linear, least-squares, optimization problem.

Graph-based approaches are considered to be superior to conventional EKF solutions [8], even though a more accurate research from the point of view of computational complexity is required in order to make them faster and thus more usable in on-line state estimation [9]. The graph technique implies that not only the latest state can be estimated, but also the previous ones, making it possible to continuously estimate the full robot trajectory.

An even wider generalization of the graph approach is the *factor-graph*,

which is a hypergraph in which edges do not incide only between two nodes, but can possibly affect many of them [10]. We can see an example of factor graph in Figure 3.1. In this example the $x_i$ , $i \in \{1, \ldots, 5\}$ represent the state variables and $f_j$, $j \in \{A, \ldots, E\}$ are the functions having as argument a subset of $\{x_1, \ldots, x_5\}$. The function that has to be optimized, in this case, is $g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3)f_D(x_3, x_4)f_E(x_3, x_5)$. This comes up to be a powerful tool in multi-sensor fusion problems, in particular it is appreciated the possibility to represent heterogeneous measurement, in the sense of number of poses effected, maintaining a quite explicit design of the graph [11].

## 3.2 Odometry estimation

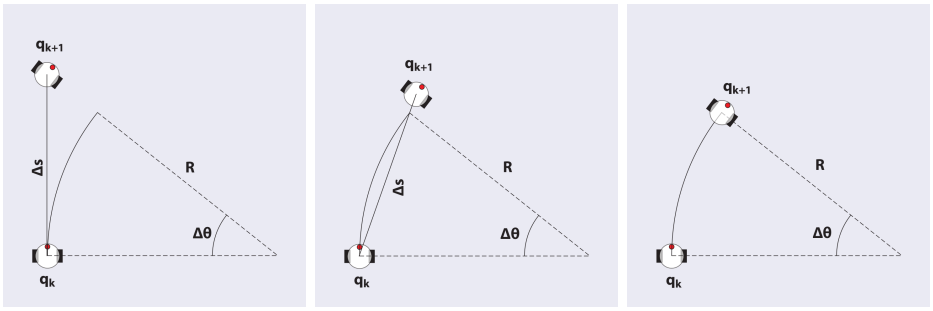
Odometry measures the distance traveled by a robot, or any kind of movable system, from an initial point, into the space in which it operates. It is crucial to have a good odometry estimation for many reasons, in particular for autonomous navigation. A good odometry estimation can be done only by retrieving and combining different sensor measurements, but certainly the information given by the wheels rotation is the basis for thee estimation for every wheeled mobile robot.

### 3.2.1 Generic odometry

As mentioned before, the wheels rotation usually gives the majority of the information about odometry, especially in the case in which an absolute pose measurement is not available, e.g., the GPS sensor is not present in indoor environment.

As we are dealing with wheeled robots we can collapse our working space in a 2D plane, to estimate its odometry means indeed to estimate the position and the orientation of the robot in a 2D space. It follows that a three element vector $(x, y, \theta)$ is enough to represent this information. More precisely, the $x$ and the $y$ represent the two coordinates on the plane, considering the origin $(0, 0)$ the initial position, and $\theta$ the rotation of the robot around its vertical axis, with the respect of the initial orientation.

In order to be able to move on a plane, each wheeled mobile robot (WMR) must have a single point around which all wheels follow a circular course. This point is known as the instantaneous center of curvature (ICC) or the instantaneous center of rotation (ICR). In practice, it is quite simple to identify because it must lie on a line coincident with the rotation axis of each wheel that is in contact with the ground. Thus, when a robot turns,




(a) Euler method (b) Runge-Kutta method (c) Exact reconstruction

*Figure 3.2: Different integration methods results*

the orientation of the wheels must be consistent and a ICC must be present otherwise the robot cannot move.

If we could retrieve the sequence of the exact position variation of the robot $(\Delta x, \Delta y, \Delta\theta)$ at a good rate, the odometry would be simply the integration of these measurement. These delta positions could be, if necessary, derived from the velocity along the axis.

Defined as $v(t)$ the linear velocity at a $t$ instant of time and $\omega(t)$ the angular velocity at the same time, we can retrieve the position and orientation of the robot at time $t_1$ as follows

$$\begin{aligned} x(t_1) &= \int_0^{t_1} v(t)cos(\theta(t))dt\,, \\ y(t_1) &= \int_0^{t_1} v(t)sin(\theta(t))dt\,, \\ \theta(t_1) &= \int_0^{t_1} \omega(t)dt\,. \end{aligned} \tag{3.2}$$

In order to deal with concrete cases, namely discrete time, different integration methods exist. We will present three among the most common ones: Euler method, II order Runge-Kutta method and the exact reconstruction [12]. For these explanation we will use the relaxed notation $x_k = x(t_k), v_k = v(t_k)$ and so on, and we will define $T_s = t_{k+1} - t_k$, namely the sampling period.

**Euler method**

$$\begin{cases} x_{k+1} = x_k + v_k T_s cos\theta_k \\ x_{k+1} = y_k + v_k T_s sin\theta_k \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases} \tag{3.3}$$

Euler method is the simplest integration method, but also the most subject to error in $x_{k+1}$ and $y_{k+1}$. $\theta_{k+1}$ is exact and it will be used also for all the other integration methods, indeed. The whole system is correct for straight path. In general, the error decreases as $T_s$ gets smaller.

**II order Runge-Kutta method**

$$\begin{cases} x_{k+1} = x_k + v_k T_s cos(\theta_k + \frac{\omega_k T_s}{2}) \\ x_{k+1} = y_k + v_k T_s sin(\theta_k + \frac{\omega_k T_s}{2}) \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases} \tag{3.4}$$

Compared with the Euler method, the II order Runge-Kutta decreases the error in computation of $x_{k+1}$ and $y_{k+1}$ using the mean value of $\theta_k$. Even in this case, the smaller is the sampling period $T_s$, the smaller is the error.

**Exact reconstruction**

$$\begin{cases} x_{k+1} = x_k + \frac{v_k}{\omega_k}(sin\theta_{k+1} - sin\theta_k) \\ x_{k+1} = y_k - \frac{v_k}{\omega_k}(cos\theta_{k+1} - cos\theta_k) \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases} \tag{3.5}$$

With the exact reconstruction we can retrieve the actual arc of circumference that the robot traveled. It is computed using the instantaneous radius of curvature $R = \frac{v_k}{\omega_k}$. Note that for $\omega_k = 0 \rightarrow R = \infty$ the equation degenerates, matching the Euler and Runge-Kutta algorithms.

### 3.2.2 Differential drive odometry

The differential drive kinematics consists in two active wheels, rotating on a common axis, driven by two different motors. In addition, one or more passive castor wheel(s) can support the robot stability without interfering with the robot kinematic. Many commercial robots adopt this kind of mechanism due to the simplicity of implementation, the relative low cost and the compactness of the system. We can find it in TurtleBot [13], Khepera [14] or Robuter [15], represented respectively in Figure 3.3(a), Figure 3.3(b) and Figure 3.3(c)

The whole robot motion is based on the difference in rotation velocity of the two active wheels. The ICC lies, as expected, on the wheels rotation

(a) TurtleBot

(b) Khepera

(c) Robuter

*Figure 3.3: Commercial differential drive robots*

axis and the curve that the robot will follow depends on the position of this point with the respect of the middle point between the two wheels, as shown in Figure 3.4. The relationship between the wheels velocity $v_r$ and $v_l$, the distance between the ICC and the midpoint of the wheels $R$, and the angular velocity of the robot $\omega$ are given by the following

$$\begin{aligned} \omega(R + \frac{l}{2}) &= v_r \,, \\ \omega(R - \frac{l}{2}) &= v_l \,, \end{aligned} \tag{3.6}$$

where $l$ is the distance between the 2 wheels.

We are actually interested in retrieving $R$ and $\omega''$ from the velocities, which are usually the available data given by the encoders, and $l$ which is a fixed parameter. So the formulas are:

$$\begin{aligned} R &= \frac{l(v_r + v_l)}{2(v_r - v_l)} \,, \\ \omega &= \frac{v_r - v_l}{l} \,. \end{aligned} \tag{3.7}$$

It is interesting to analyze a couple of special cases. When $v_r = v_l$ then $R = \infty$, which means that the robot is moving in a straight line. When

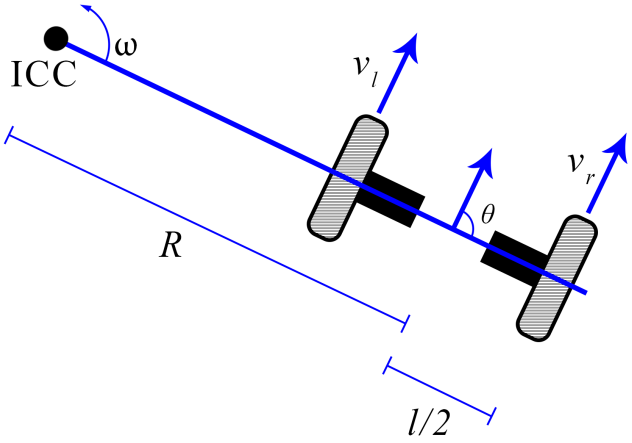



*Figure 3.4: Differential drive kinematics*

$v_l = -vr$ then $R = 0$, which means that the robot is only rotating around the vertical axis passing through the midpoint between the wheels.

Since the differential drive structure is non-holonomic, the robot is not able, for example, to perform a lateral movement or any other kind of displacement not represented by the previous equations. The position $(x', y', \theta')$ at a particular instant of time is given by the computation on the previous $(x, y, \theta)$ and the time span $\Delta t$ between the two positions

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(\omega\Delta t) & -sin(\omega\Delta t) & 0 \\ sin(\omega\Delta t) & cos(\omega\Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\Delta t \end{bmatrix}, \quad (3.8)$$

where $ICC_x$ and $ICC_y$ are the coordinates computed as it follows:

$$ICC = (x - Rsin(\theta), y + Rcos(\theta))$$

The specific case for the odometry calculation in differential drive, according with 3.2 is

$$\begin{aligned} x(t_1) &= \frac{1}{2}\int_0^t (v_r(t) + v_l(t))cos(\theta(t))dt \\ y(t_1) &= \frac{1}{2}\int_0^t (v_r(t) + v_l(t))sin(\theta(t))dt \\ \theta(t) &= \frac{1}{l}\int_0^t (v_r(t) - v_l(t))dt \end{aligned} \quad (3.9)$$

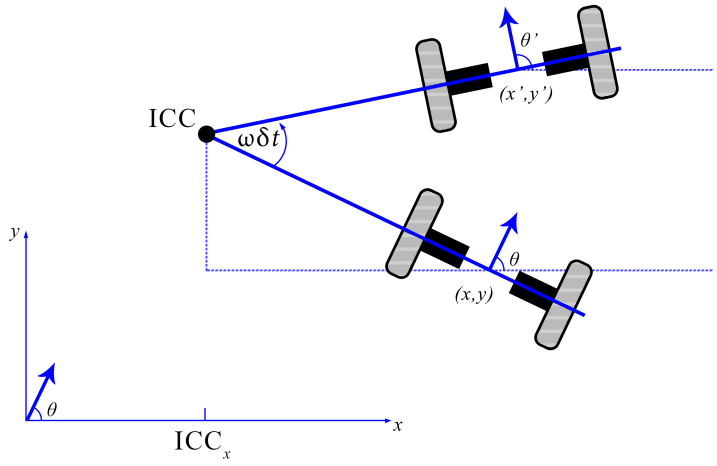



*Figure 3.5: Differential drive robot motion from pose* $(x, y, \theta)$ *to* $(x', y', \theta')$

### 3.2.3 Skid-steering odometry

The skid-steering mechanics tries to maintain the simplicity and compactness of the differential drive model, improving in the meantime its robustness. It consists in four wheels that we can consider separately as a pair on the left, front and rear, and a pair on the right, again front and rear. Each couple of wheels is driven by one motor, located in the middle of the front and rear wheel. These two wheels are usually connected by a transmission belt or a chain. Each motor allows to rotate the pair of wheels connected at with the same velocity.

Skid-steering compactness and the high maneuverability [16][17], in addition to the higher robustness, with the respect of the differential drive, make this kind of mechanic an optimal choice for different purposes [18][19]. This mechanics, indeed, offers a good mobility on different terrains, not only indoor, but also outdoor ones, because one of its advantages is the possibility of installing tracks for the terrains that require them, without changing the whole mechanic.

Unfortunately, the drawback is a more complex kinematics because the pure rolling and no-slip assumption - which was possible to use for the differential drive case - is no more an option: the wheels *must* slip during a curve. This implies a hard-to-predict motion, given the velocity input. Other disadvantages are an energy inefficient motion and a fast tires' consumption, caused by the slippage needed for curving [20].

Wang et al. [21] help to partially resolve the problem of the complex skid-steering kinematic proposing an approximation to the differential drive

*Figure 3.6: Ra.Ro. wheels*

kinematic based on three assumptions:

(i) the mass center of the robot is located at the geometric center of the body frame

(ii) the two wheels of each side rotate at the same speed ($w_{fr} = w_{rr}$ and $w_{fl} = w_{rl}$)

(iii) the robot is moving on a firm ground surface, and all the four wheels are always in contact with the ground surface

Then, considering the Figure 3.7 and the deriving the Equations 3.9 we obtain

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = f \begin{bmatrix} w_l r \\ w_r r \end{bmatrix} \tag{3.10}$$

where $v = (v_x, v_y)$ represents the vehicle's translational velocity with respect to its local frame, $w_z$ represents its angular velocity, $r$ represents the radius of the wheels and $w_l$ and $w_r$ represent respectively the angular velocity of the left and right wheels.

While the robot is curving there are different $ICR$: $ICR_l$, $ICR_r$ and $ICR_G$, that belongs respectively of the left-side tread, right-side tread, and the robot center of mass, as shown in Figure 3.5. We define the coordinates of the $ICRs$ respect to the local frame as $(x_l, y_l)$, $(x_r, y_r)$ and $(x_G, y_G)$. All of the treads share the same angular velocity $\omega_z$, so these equations follows

$$y_G = \frac{v_x}{w_z}\,, \tag{3.11}$$

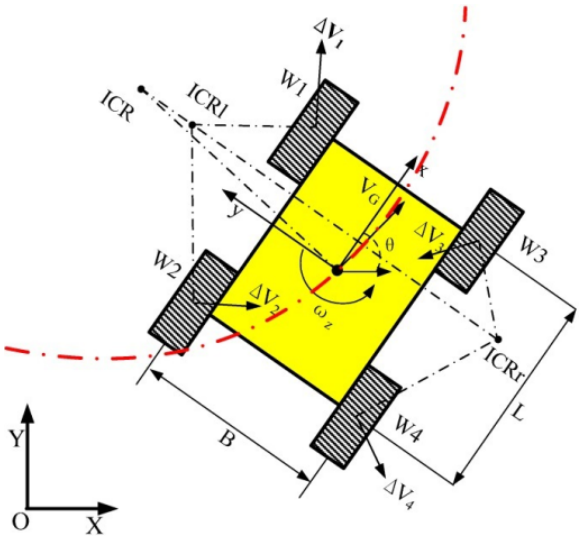



*Figure 3.7: Skid-steering platform*

$$y_l = \frac{v_x - w_l r}{w_z}\,, \tag{3.12}$$

$$y_r = \frac{v_x - w_r r}{w_z}\,, \tag{3.13}$$

$$x_G = x_l = x_r = -\frac{v_y}{w_z}\,. \tag{3.14}$$

From (3.9) to (3.14) the generic odometry kinematic (3.2) can be represented as:

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = J_w \begin{bmatrix} w_l r \\ w_r r \end{bmatrix}\,. \tag{3.15}$$

where $J_w$ depends on $IRCs$ coordinates and is defined as follows:

$$J_w = \frac{1}{y_l - y_r} \begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & 1 \end{bmatrix}\,. \tag{3.16}$$

When the robot is symmetrical, then the $ICRs$ lies symmetrically on the $x$-axis, and we have $x_G = 0$ and $y_0 = y_l = -yr$. The $J_w$ matrix can be rewritten as:

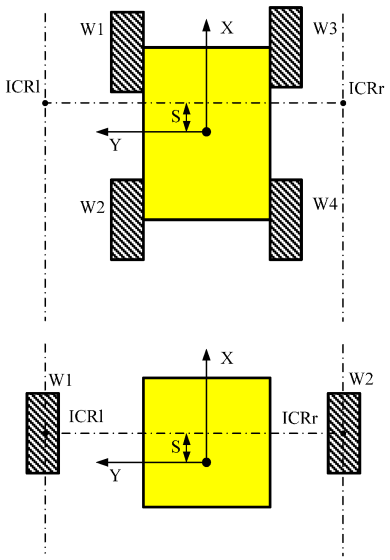



*Figure 3.8: Geometric equivalence between the wheeled skid-steering robot and the ideal differential drive robot*

$$J_w = \frac{1}{2y_0} \begin{bmatrix} y_0 & y_0 \\ 0 & 0 \\ -1 & 1 \end{bmatrix} . \tag{3.17}$$

So the velocities defined in Equations 3.15 can be defined, for the symmetrical model, as follows:

$$\begin{cases} v_x = \dfrac{v_l + v_r}{2} \\ v_y = 0 \\ w_z = \dfrac{-v_l + v_r}{2y_0} \end{cases} \tag{3.18}$$

and from Equation 3.7 we can get the instantaneous radius of the path curvature:

$$R = \frac{v_G}{w_z} = \frac{v_l + v_r}{-v_l + v_r} y_0 \, . \tag{3.19}$$

The ratio between the sum and difference of left and right wheels linear velocities [22] can be defined as a variable $\lambda$:

$$\lambda = \frac{v_l + v_r}{-v_l + v_r} \, , \tag{3.20}$$

and Equation 3.19 becomes:

$$R = \lambda y_0 \, . \tag{3.21}$$

A similar approach is used in Mandow's work [23], in which an $IRC$ coefficient $\chi$ is defined as:

$$\chi = \frac{y_l - y_r}{B} = \frac{2y_0}{B}, \qquad \chi \geq 1 \tag{3.22}$$

$\chi$ represents the approximation from the differential drive kinematic.

We can notice that when $\chi$ is equal to 1 there is no slippage and the skid-steering model coincides with the differential drive. It implies that we can approximate the skid-steering model as a differential drive model, working on the $\chi$ variable. In particular, the skid-steering coincides with a differential drive with a larger span between the left and right wheels as shown in Figure 3.8. This is a very useful method and it will be very useful for our work.

## 3.3 Sensor fusion framework

As mentioned before, pure kinematics equations are actually just an approximation of the real world, they are not sufficient to retrieve the correct odometry of a robot with enough precision. This is why multi-sensor fusion is required. Here we introduce the framework used to make this process in a way possible to set up.

### 3.3.1 ROAMFREE

The acronym ROAMFREE stands for Robust Odometry Applying Multi-sensor Fusion to Reduce Estimation Errors. The main aim of the framework is to offer a set of mathematical techniques and to perform sensor fusion in mobile robotics, focusing on pose tracking and parameter self-calibration [24]. The main goals of the ROAMFREE project include ensuring that the resulting software framework can be employed on very different robotic platforms and hardware sensor configurations and easily tuning to specific user needs by replacing or extending its main components.

In ROAMFREE the information fusion problem is formulated as a fixed-lag smoother whose goal is to track not only the most recent pose, but all the positions and attitudes of the mobile robot in a fixed time window: short lags allow real time pose tracking, still enhancing robustness with respect to measurement outliers; longer lags allow for online calibration, where the goal is to refine the available sensor parameters estimation.

The system is based on a graph-based approach. In particular, a factor graph is generated; this graph keeps the probabilistic representation of the

pose retrieved by the sensor fusion measurements, the estimated sensor parameters and the sensor error models. All the modules interact in some way with this graph, for example to update it with new measurements or new estimated poses. The factor graph is designed to allow an arbitrary number of sensors, even if they work at different rates, without a predictable rate or producing obsolete data.

The framework implements a set of *logical* sensors, which are independent from the actual hardware that produces the measurement. For example, an odometry measurement can be retrieved from a laser scanner elaboration or a wheels' encoders one, but both can be properly setted up as a logical *GenericOdometer* sensor. Each logical sensor is characterized by a parametric error model specific to its domain. This means that we must initialize the sensors passing the specific parameters for that sensor. For example, a *DifferentialDriveOdometer* requires the wheels radii and the wheels distance, expressed in meters. Another required parameter to properly set up a sensor is its position and orientation with the respect to the tracked base frame. For example, the camera sensor, possibly used to retrieve markers positions, must be properly set in order to calculate the exact transformation between the marker seen and the base frame.

ROAMFREE's modularity and its ROS implementation make it a very powerful tool for sensor fusion purposes. Unfortunately we had to deal with a lack of documentation which made the development of a stable ROS node quite hard.

**Factor graph filter** As mentioned before, the core of ROAMFREE lies in the factor graph filter. We have already written about the important features of graph-based approaches in Subsection 3.1.2 in a general way. The most relevant advantage is the possibility of manage high-dimensional problems in a relatively short time. The graph holds the full joint probability of sensor readings given the current estimate of state variables, representing its factorization in terms of single measurement likelihoods. Each node of the graph contains a the pose of the robot and the sensors' calibration parameters, i.e. gains, biases, displacement or misalignments. The nodes are generated by new measurements, represented as hyper-edges (*factors*) connecting one or more nodes, depending on their order. For example, a velocity measurement connects two nodes since it needs one integration to retrieve a position; an acceleration needs three nodes because two integrations are required.

We need to choose one, and only one, of the available sensors as an *architecture master sensor*, and a good practice is to choose an odometry

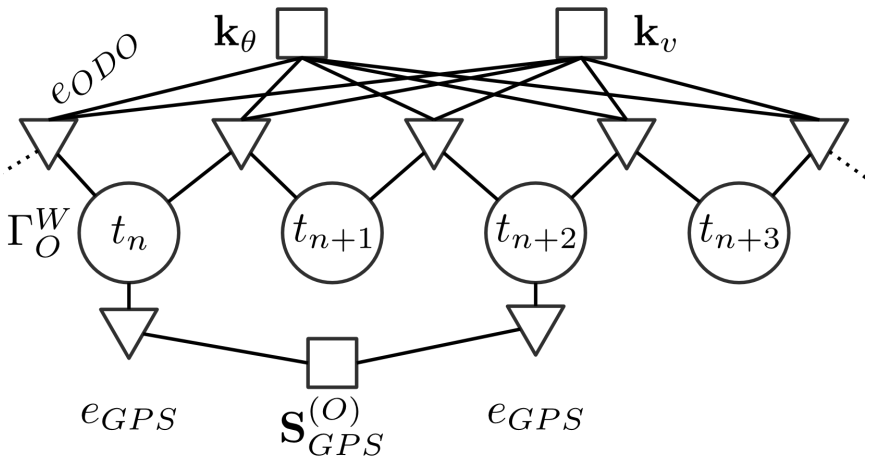



*Figure 3.9: An instance of the pose tracking factor graph with four pose vertices $\Gamma_O^W(t)$ (circles), odometry edges $e_{ODO}$ (triangles), two shared calibration parameters vertices $k_\theta$ and $k_v$ (squares), two GPS edges $e_{GPS}$ and the GPS displacement parameter $\mathbf{S}_{GPS}^{(O)}$*

sensor to play this role, because it usually has a high rate and it is the one that gives a hopefully good starting point for the optimization process. Once the master sensor measurement is collected, an initial guess of the new pose is made and then a non-linear optimization process begins, also using the other measurements retrieved in the meantime. It can happen that sensor readings are late for low rate, connection problems or in general are not available. In these cases, if the available ones are not sufficient to generate a pose, the measurement handling is delayed until enough data are available.

Node generation is based on a fixed-lag window. It means that only the nodes contained in this time span are considered for the following pose. Older nodes and factors, since are no more used, are deleted. In order to avoid high loss of information older nodes and factors can be marginalized, keeping the information they used to hold in a new generated factor.

We can resume the factor-graph advantages here:

- Flexible with respect to sensor nature and number. The modularity of the system allows to manage all the inserted sensors in an independent and uniform way, by means of the abstract hyper-edges interface, as they are inserted into the graph.

- Sensors can be, if necessary, turned on and off during the process. The factors management is asynchronous, so they can be added into the graph as soon as new readings are available.

- It is possible to deal with out-of-order measurements. If an old infor-

mation is received, according to its time stamp, it is not discarded, but an appropriate factor is created, connecting the nodes interested by it and consequentially updating and refining them.

- The resulting estimation is higher in quality and, in certain circumstances, i.e., with many state variables faster than traditional filters, such as EKFs [8].
- The high degree of sparsity of the considered information fusion problem is explicitly represented and it can be exploited by inference algorithms. Indeed, in our case a factor may involve up to three robot poses; moreover, it is difficult to imagine a robot employing much more than ten sensors for pose estimation, implying that each pose is incident to a limited number of factors.

**Error models** For each logical sensor model an error model definition is needed. All of theme come a common definition

$$e_i(t) = \hat{z}(t; \hat{x}_{S_i}(t), \xi) - z + \eta\,, \tag{3.23}$$

where $\hat{x}_{S_i}(t)$ represents the extended state for the sensor frame $S_i$ in which the position and the orientation of the frame with the respect of the world frame at time $t$; $\xi$ represents the vector of the parameters relative to that sensor and $\hat{z}(t; \hat{x}_{S_i}(t), \xi))$ is a predictor measurement computed as a function of the previously defined parameters and the incident nodes. $z$ is the real sensor output and $\eta$ is a zero-mean Gaussian noise representing the measurement uncertainty. It is evident that the more the prediction is accurate, the more the error is near to 0, net of the Gaussian noise.

The equation that describes the actual error depends on the type of measurement considered. We can identify five classes:

(i) absolute position and/or orientation (e.g., GPS)

(ii) linear and/or angular velocity in sensor frame (e.g., gyroscope)

(iii) acceleration in sensor frame (e.g., accelerometer)

(iv) vector field in sensor frame (e.g., magnetometer)

(v) landmark pose with respect to sensor (e.g., markers)

Moreover, the other thing that characterize the sensor in ROAMFREE is the parameter's vector $\xi$, which includes gain, bias and other specific parameters according to the sensor class. These parameters sometimes are

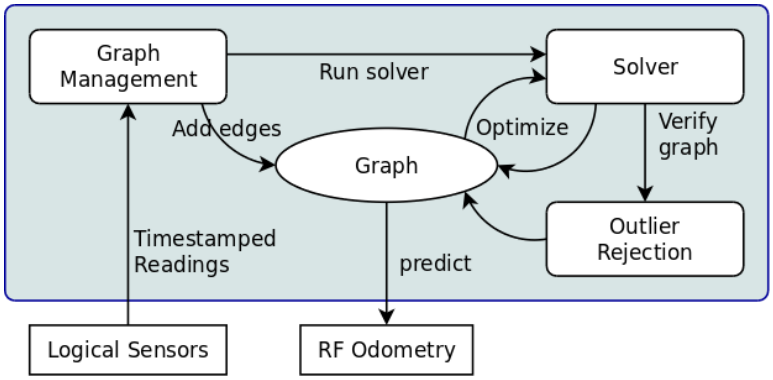



*Figure 3.10: ROAMFREE estimation schema*

easy to retrieve from sensors specifications or from observation, but they often need an accurate tuning to let everything work properly.

Every time a measurement is added to the factor-graph we need to pass as parameter also the covariance matrix ,in addition to the measurement itself and the sensor name.. Usually, in ROAMFREE, is used a diagonal matrix, with the measurement variances on the diagonal, and zeros otherwise. We must be careful in covariance matrix setup. Changing the value of the variances means to change the reliability of a sensor. In other words, if two measurements indicate two conflicting outputs, the system will *trust* more the one with a lower covariance. The covariance matrix can be even different according to the actual measurement, of course always being of the right dimension.

A convenient feature is the outlier management made through the *robust kernel* technique. It consists in setting a threshold in the measurement domain. If this threshold is exceeded in module, the error model of the sensor, for that measurement, becomes linear instead of quadratic, which means that it is less involved in the subsequent computation. This is useful to deal with errors that can sometimes occur in data retrieving.

**Optimizations** The optimization algorithms implemented in ROAMFREE are Gauss-Newton and Levenberg-Marquardt. Both of them require a problem formulated as a non-linear, wighted and least-squares optimization and here we will discuss about how this is done. Considering the error function $e_i(x_i, \eta)$ associated to the $i$-th edge of the hyper-graph and defined as (3.23). We can approximate the error function as $\bar{e}_i(x_i) = e_i(x_i, \eta)|_{\eta=0}$ since $e_i$ is a random vector. It can involve non-linear dependencies with respect to the noise, so its covariance $\Sigma_\eta$ is computed by means of linearization, i.e.:

$$\Sigma_{e_i} = J_{i,\eta}\Sigma_\eta J_{i,\eta}^T|_{x_i=\breve{x}_i,\eta=0} \tag{3.24}$$

where $J_{i,\eta}$ is the Jacobian of $e_i$ with respect to $\eta$ evaluated in $\eta = 0$ and in the current estimate $\breve{x}_i$. The covariance matrix $\Sigma_\eta$ is the one we mentioned before and here is where it is involved in the optimization.

A negative log-likelihood function can be associated to each edge in the graph, which stems from the assumption that zero-mean Gaussian, noise corrupts the sensor readings. Omitting the terms which do not depend on $x_i$, for the $i$-th edge this function reads as:

$$\mathcal{L}_i(x_i) = \overline{e}_i(x_i)\Omega_{e_i}\overline{e}_i(x_i) \tag{3.25}$$

where $\Omega_{e_i} = \Sigma_{e_i}^{-1}$ is the information matrix of the $i$-th edge. The solution to the information fusion problem is given by the assignment for the state variables such that the likelihood of the observations is maximum.

$$\mathcal{P} = \underset{x}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}_i(x_i) \tag{3.26}$$

We can observe that this is a non-linear least-squares problem where the weights are the information matrices associated to each factor. If a reasonable initial guess for $x$ is known, a numerical solution for $\mathcal{P}$ can be found by means of the popular Gauss-Newton (GN) or Levenberg-Marquardt (LM) algorithms. A complete schema of the interaction among the ROAMFREE modules is represented in Figure 3.10.

## 3.4 Simultaneous Localization and Mapping

With simultaneous localization and mapping (SLAM) is meant the building of a map while the robot locates itself into the map that is being created. The SLAM can be considered as a preliminary phase in which the robot creates the map which it will use for the autonomous navigation, or it can be contextual to the autonomous navigation. It is important to point out that it does not matter whether the robot, during the SLAM phase, is human-controlled or not.

Although the first localization guess is given by the odometry, as mentioned in Section 3.2, it accumulates error as long as the robot moves. A good odometry estimation is desirable for the SLAM problem, but the error given by the odometry can be corrected by using world references observed through laser scanner(s), camera(s) (Visual SLAM) or similar sensors. It follows that the observation should be done in an environment as static as

possible, even if SLAM frameworks can deal with mobile objects. Moreover, the localization makes sense if it is made with respect of a map, but if the map is being made at the same time is evident the possible problem that can occur. SLAM must be done in a recursive way and this is one of the main reasons why it is such a complex task.

From a probabilistic point of view, there are two main forms of SLAM. One is known as the online SLAM problem algorithm which involves estimating the posterior over the momentary pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \,, \tag{3.27}$$

Here $x_t$ is the pose at time $t$, $m$ is the map and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls, respectively. This problem only involves the estimation of the variables that exist at time $t$. Algorithms for the online SLAM problem are incremental: they discard past measurements and controls once they are processed. The second SLAM problem is called the full SLAM problem. Here we want to calculate a posterior over the entire path $x_{1:t}$ along with the map, instead of just the current pose $x_t$:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.28}$$

Instead of incrementally computing the state as in the online case, here the sequence of states is computed once.

Regardless of the method used to implement a full or online SLAM, the algorithm must follow these steps:

(i) Landmarks detection: the robot must recognize some features from the environment, called landmarks. Considering a 2D map, horizontal LIDAR are the most common sensor used for this kind of task. Angles, edges, particular shapes are good candidates to be detected as a landmark.

(ii) Data association: once the landmark is been detected, it must be matched with a possibly existing landmark into the map. It can be a hard task because a single feature can match with many, growing exponentially as long as the map grows.

(iii) State estimation. It uses observations and odometry to reduce errors. The convergence, accuracy, and consistency of the state estimation are the most important properties. Thus, the SLAM method must maintain the robot path and use the landmarks to extract metric constraints to compensate the odometer error.

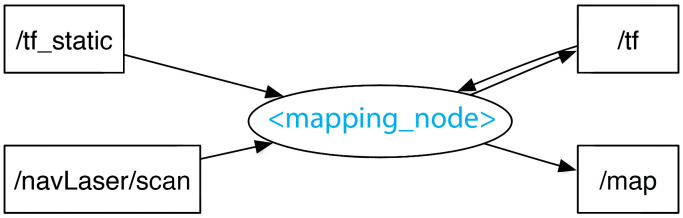



*Figure 3.11: The image represents the topics subscribed and published by the mapping node, independently of the exact mapping system used*

The major difficulties of SLAM are the following:

- **High dimensionality**: since the map dimension constantly grows when the robot explores the environment, the memory requirements and time processing of the state estimation increase. Some submapping techniques can be used to reduce these consumption, at the cost of a worse performance.

- **Loop closure**: when the robot returns to a place it has been previously, the accumulated odometry error might be large. Then, the data association and landmark detection must be effective in correcting the odometry. Place recognition techniques are used to cope with the loop closure problem.

- **Dynamics in environment**: state estimation and data association can be confused by the inconsistent measurements in the dynamic environment. There are some methods that try to deal with these environments.

We will focus on 2D SLAM, using a laser scanner as a sensor and related laser scans measurements. In general, the aim of a SLAM framework is to collect the laser scans and try to associate them in one occupancy grid map. Then, if following scans match with the memorized map, the position will be corrected according to this match, hopefully improving the localization estimation. The more various the environment is, the more the localization is easy to make, because the probability to deal with a potential ambiguity is lower.

**Algorithm 2** Improved RBPF for map learning

**Require:**
$\mathcal{S}_{t-1}$, the sample set of the previous time step
$z_t$, the most recent laser scan
$u_{t-1}$, the most recent odometry measurement
**Ensure:**
$\mathcal{S}_t$, the new sample set
$\mathcal{S}_t = \{\ \}$
**for all** $s_{t-1}^{(i)} \in \mathcal{S}_{t-1}$ **do**
  $\langle x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} \rangle = s_{t-1}^{(i)}$
  *//Scan − matching*
  $x_t^{'(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$
  $\hat{x}_t^{(i)} = \text{argmax}_x\, p(x|m_{t-1}^{(i)}, z_t, x_t^{'(i)})$
  **if** $\hat{x}_t^{(i)} = \text{failure}$ **then**
    $x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    $w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t|m_{t-1}^{(i)}, z_t, x_t^{'(i)})$
  **else**
    *//Sample around the mode*
    **for all** $k = 1$ to $K$ **do**
      $x_k \sim \{x_j \mid |x_j - \hat{x}^{(i)}| < \Delta\}$
    **end for**
    *//Compute Gaussian proposal*
    $\mu_t^{(i)} = (0, 0, 0)^T$
    $\eta^{(i)} = 0$
    **for all** $x_j \in \{x_1, \ldots, x_K\}$ **do**
      $\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
      $\eta^{(i)} = \eta^{(i)} + p(z_t|m_{t-1}^{(i)}) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    **end for**
    $\mu_t^{(i)} = \mu_t^{(i)} / \eta^{(i)}$
    $\Sigma_t^{(i)} = \mathbf{0}$
    **for all** $x_j \in \{x_1, \ldots, x_K\}$ **do**
      $\Sigma_t^{(i)} = \Sigma_t^{(i)} + (x_j - \mu_t^{(i)})(x_j - \mu_t^{(i)})^T \cdot p(z_t|m_{t-1}^{(i)} \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
    **end for**
    $\Sigma_t^{(i)} = \Sigma_t^{(i)} / \eta^{(i)}$
    *//Sample new pose*
    $x_t^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$
    *//Update importance weights*
    $w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}$
  **end if**
  *// Update map*
  $m_t^{(i)} = \text{integrateScan}(m_{t-1}^{(i)}, x_t^{(i)}, z_t)$
  *// Update sample set*
  $\mathcal{S}_t = \mathcal{S}_t \cup \{\langle x_t^{(i)}, w_t^{(i)}, m_t^{(i)} \rangle\}$
**end for**
$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}$
**if** $N_{eff} < T$ **then**
  $\mathcal{S}_t = \text{resample}(\mathcal{S}_t)$
**end if**

### 3.4.1 Gmapping

`gmapping` is the most widely used laser-based SLAM package in robotic field, worldwide. The algorithm was proposed by Grisetti et al. in 2007 and it is a Rao-Blackwellized Particle Filter SLAM approach [25].

We mentioned the Particle Filter in Subsection 3.1.1 and here we will describe RBPF, which is an optimized version for the SLAM problems. Let's start from (3.28) and factorize it as:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(m|x_{1:t}, z_{1:t}) p(x_{1:t}|z_{1:t}, u_{1:t-1})\,, \tag{3.29}$$

this factorization allows to first estimate only the trajectory of the robot and then to compute the map given that trajectory. In particular, $p(m|x_{1:t}, z_{1:t})$ can be easily computed using "mapping with known poses" since $x_{1:t}$ and $z_{1:t}$ are known.

If new control data $u_t$ from the odometry and a new measurement $z_t$ form the laser scanner is available; the RBPF occupancy grid SLAM works as follows.

1. It determines the initial guess $x_t^{'(i)}$, based on $u_t$ and the pose, since the last filter $t$ update $x_{t-1}$ has been estimated.

2. It performs a scan matching algorithm based on the map $m_{t-1}^{(i)}$ and $x_t^{'(i)}$. If the scan matching fails, the pose $x_t^{(i)}$ of particle $i$ will be determined according to a motion model, otherwise the next two steps will be performed.

3. If the scan matching is successfully done, a set of sampling points around the estimated pose $\hat{x}_t^{(i)}$ of the scan matching will be selected. Based on this set of $t$ poses, the proposal distribution will be estimated.

4. It draws pose $x_t^{(i)}$ of particle $i$ from the approximated Gaussian distribution of the improved proposal distribution.

5. It performs an update of the importance weights.

6. It updates map $m^{(i)}$ of particle $i$ according to $x^{(i)}$ and $z_i$.

The more detailed RBPF algorithm pseudo-code can be read in Algorithm 2. The author proposes a way to compute an accurate distribution by taking into account both the movement of the robot and the most recent observations. This decreases the uncertainty about the robot's pose in the prediction step of the particle filter. As a consequence, the number of particles required decreases since the uncertainty is lower, due to the scan matching process, improving the performance.

### 3.4.2 Cartographer

Another possible approach to the SLAM problem is using graph-based methods. These methods use optimization techniques similar to the factor graph previously introduced to transform the SLAM problem into a quadratic programming problem. The historical development of this paradigm has been focused on pose-only approaches using the landmark positions to obtain constraints for the robot path. The objective function to optimize is obtained

assuming Gaussianity. Since these methods are based on a factor graph, they are able to better remember the previous sub-maps and the previous localization and thus they prove to be more accurate with respect to other approaches. On the other hand, their main disadvantage is the high computational time they take to solve the problem, for this reason they are usually suitable to build maps off-line.

Google's Cartographer provides a real-time solution to indoor and outdoor mapping. The system generates submaps from the matching of the most recent scans at the best estimation position, which is assumed to be accurate enough for short periods of time. Since the scan matching only works on submaps, the error of the pose estimation in the world frame eventually increases. For this reason, the system runs periodically a pose optimization algorithm. When a submap is considered to be finished, no more scans are added to it and it takes part in scan matching for loop closure. If the robot estimated pose is close enough to one or more processed submaps, the algorithm runs the scan matching between the incoming laser scans and those maps. If a good match is found, it is added as a loop closing constraint to the optimization problem. By completing the optimization every few seconds, the loops are closed immediately once a location is revisited. This leads to the soft real-time constraint that the loop closure scan-matching has to happen before the new scans are added, otherwise it will fall behind noticeably. This has been achieved by using a branch-and-bound approach and several precomputed grids per finished submap.

## 3.5 Localization

As mentioned before, robot localization is the problem of estimating a robot pose relative to the map of the operational environment. It has been defined as one of the most fundamental problems in mobile robotics [26].

We can recognize different levels of localization problems. The localization tracking is the simplest one; the robot starts from a known position and the localization aim is to correct the hopefully small odometry errors. A more challenging problem is the global localization problem; the robot must localize itself without a given initial position. An even more difficult problem is the *kidnapped robot* problem [27]; it can happen when a localized robot is moved, with no information about this transportation, to a different location. It might seem a similar problem to the second one, but here we can not trust a measurement as consistent with a previous one, because of the loss of information during the unexpected movement.

**Algorithm 3** Adaptive variant of Monte Carlo Localization

1: **procedure** AMCL($\mathcal{X}_{t-1}, u_t, z_t, m$)
2: $\overline{\mathcal{X}}_t = \mathcal{X}_t = 0$
3: **for all** $m := 1$ to$M$ **do**
4: $x_t^{(m)} = \text{SampleMotionModelOdometry}(u_t, x_{t-1}^{(m)})$
5: $w_t^{(m)} = \text{MeasurementModel}(z_t, x_t^{(m)}, m)$;
6: $\mathcal{X}_t = \mathcal{X}_t - \langle x_t^{(m)}, w_t^{(m)} \rangle$
7: $w_{avg} = w_{avg} + \frac{1}{M} w_t^{(m)}$
8: **end for**
9: $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
10: $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
11: **for all** $m := 1$ to $M$ **do**
12: **with probability** max(0.0,1.0 - $\frac{w_{fast}}{w_{slow}}$) **do**
13: add random pose to $\mathcal{X}_t$
14: **else**
15: draw $i \in \{1, \ldots, N\}$ with probability $\propto x_t^{(m)}$
16: add $x_t^{(i)}$ to $\mathcal{X}_t$
17: **end with**
18: **end for**
19: return $\mathcal{X}_t$
20: **end procedure**

In other words, the localization problem consists in identifying an appropriate coordinate transformation between the global frame, which is fixed and integral with the map, and the robot frame. Then, a detected object from the robot's point of view can be, in turn, transformed with the respect of the global frame by coordinate transformation.

In robot localization the state $x_t$ of the system is the robot pose, which for the two dimensional mapping, is typically represented as a three dimensional vector $x_t = (x, y, \theta)$ in which $x$ and $y$ indicate the position of the robot in the map plane, and $\theta$ the angle formed by the robot orientation. The state transition probability $p(x_t|x_{t-1}, u_{t-1})$ describes how the robot position changes given the previous position $x_{t-1}$ and the new sensors' measurements $u_{t-1}$. The perceptual model $p(z_t|x_t)$ describes the likelihood of making the observation $z_t$ given that the robot is at position $x_t$.

### 3.5.1 Adaptive Monte Carlo Localization (AMCL)

The Adaptive Monte Carlo Localization (AMCL) is a method to localize a robot in a given map. It is an improved implementation of a particle filter. The word "adaptive" means that the number of particle used for the Monte Carlo localization is not fixed, but changes according to the situation. This number of particles is retrieved using the KLD-Sampling (Kulback-Leibler-Divergence) [28] [29]. The AMCL pseudocode algorithm is reported in Algorihm 3. It requires the set of particles of the last known state $\mathcal{X}_{t-1}$ and the control data $u_t$ for the prediction; the measurement data $z_t$ and the map $m$ for the update.

The algorithm returns the new state estimation as a set of particles $\mathcal{X}_t$. This filter implementation is able to deal with the global localization problem, the localization tracking and the kidnapping problem. The AMCL is flexible with the respect of the resampling technique, that means that an arbitrary one can be used. Another advantage is that AMCL is able to recover from localization errors by adding some random particles to the $\mathcal{X}_t$ set, after a specified decade (lines 15 and 16 of 3). An AMCL ROS package is available [30] and a lot of robots use this package for the localization since it provides a good configuration parameter suite.

## 3.6 A note on ROS reference system

Developers of drivers, models, and libraries need a share convention for coordinate frames in order to better integrate and re-use software components. Shared conventions for coordinate frames provide a specification for developers creating drivers and models for mobile bases. Similarly, developers creating libraries and applications can more easily use their software with a variety of mobile bases that are compatible with this specification. In this chapter we will explain the reference frames that should be used for a localization system, according to the ROS standard [31].

#### Coordinate frames

**base_link** The coordinate frame called `base_link` is rigidly attached to the mobile robot base. The `base_link` can be attached to the base in any arbitrary position or orientation; for every hardware platform there will be a different place on the base that provides an obvious reference point. A right-handed chirality with `x` forward, `y` left and `z` up is preferred.


*Figure 3.12: The tree frame representation.*

**odom** The coordinate frame called `odom` is a world-fixed frame. The pose of a mobile platform in the odom frame can drift over time, without any bounds. This drift makes the odom frame useless as a long-term global reference. However, the pose of a robot in the odom frame is guaranteed to be continuous, meaning that the pose of a mobile platform in the odom frame always evolves in a smooth way, without discrete jumps. In a typical setup the odom frame is computed based on an odometry source, such as wheel odometry, visual odometry or an inertial measurement unit. The odom frame is useful as an accurate, short-term local reference.

**map** The coordinate frame called `map` is a world fixed frame, with its Z-axis pointing upwards. The pose of a mobile platform, relative to the map frame, should not significantly drift over time. The map frame is not continuous, meaning the pose of a mobile platform in the map frame can change in discrete jumps at any time. In a typical setup, a localization component constantly re-computes the robot pose in the map frame based on sensor observations, therefore eliminating drift, but causing discrete jumps when new sensor information arrives. The map frame is useful as a long-term global reference, but discrete jumps in position estimators make it a poor reference frame for local sensing and acting.

**earth** The coordinate frame called `earth` is the origin of ECEF (earth-centered, earth-fixed) [32]. This frame is designed to allow the interaction of multiple robots in different map frames. If the application only needs one map the earth coordinate frame is not expected to be present.

### Relationship between Frames

The relationship between coordinate frames in a robot system can be represented as a tree since each coordinate frame can have a parent coordinate frame and an arbitrary number of child coordinate frames. Thus, the frames described before are attached as represented int Figure 3.12.

The `map` frame is the parent of `odom`, and `odom` is the parent of `base_link`. Although intuition would say that both `map` and `odom` should be attached to `base_link`, this is not allowed because each frame can only have one parent in ROS implementation.

### Frame Authorities

The transform from `odom` to `base_link` is computed and broadcast by one of the odometry sources, while the transform from map to `base_link` is computed by a localization component. However, the localization component does not broadcast the transform from `map` to `base_link`. Instead, it first receives the transform from `odom` to `base_link`, and uses this information to broadcast the transform from `map` to `odom`.

The transform from earth to map is statically published and configured by the choice of map frame. If not specifically configured a fallback position is to use the initial position of the vehicle as the origin of the map frame. If the map is not georeferenced so as to support a simple static transform the localization module can follow the same procedure as for publishing the estimated offset from the map to the odom frame to publish the transform from earth to map frame.

# Chapter 4

# A new navigation system for the Ra.Ro.

*"Guybrush: Van Winslow, head to Isle of Ewe!*
*Van Winslow: Please, sir, I think we should hit land first!*
*Guybrush: Isle of Ewe... It sounds like "I Love You". Nice joke.*
*Van Winslow: [Disappointedly] Yes, sir, joke..."*

Tales of Monkey Island - The Siege of Spinner Cay

## 4.1 Navigation system overview

As we mentioned before, Ra.Ro. has already a sort of "autonomous" navigation mode. It is based on line following and visual markers indicating "turn left", "keep right", "follow me" and so on. The system is quite reliable, if we accept the fact that we must attach in some way markers on walls and/or lines on the floor, and we are sure that the robot will not deal with movable or unpredicted obstacles, but it is very far from a real autonomous navigation. It is not possible, for instance, to indicate any point on a map and expect that the robot will reach that point.

Our aim was to propose an autonomous navigation solution at least reliable as the previous one, but more powerful. Ra.Ro. is ROS based, so the most logic approach was to base on the ROS navigation stack, and to work around it. The best advantage from this approach is the modularity of the system, which modules and their interactions are represented in Figure 4.1. Here follows the explanation of the modules we used in our project:

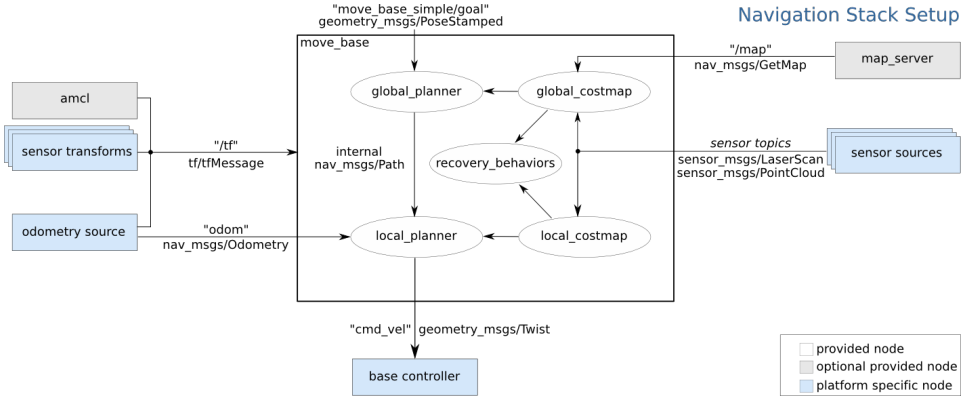



*Figure 4.1: The ROS standard navigation stack schema*

**Odometry source** The odometry source provides the estimated robot position with respect to the starting pose. The easiest way to provide this data is to use the wheel encoders, but it is usually very imprecise because of wheels slippage, different floors friction, small obstacles that let the wheel rotate without robot movement or imprecise sensor itself. To to have a better odometry source is a common choice to use a multi-sensor fusion system, and this has been the contribution of this thesis. We compared a custom source developed integrating gyroscope and raw odometry provided by the wheels encoders and two factor graph filters built using the ROAMFREE framework; the first one using IMU sensor and encoders, and the second one using, in addition, visual markers detected in the environment.

**Sensor sources** The sensor source is used by the navigation stack for mapping and for localization inside the mapped environment. It must generate PointCloud or LaserScan messages. In our case, since we mount a Hokuyo Laser scanner, we used this as sensor source. Future work on this project could add as sensor source a module that can extrapolate point cloud from cameras, but this was out of the scope of the work.

**Sensor transforms** For each sensor it is necessary to provide a transform between the base frame and the sensor frame itself. The transformation must be published as a `tf` message and, in our case, it is the static transform between the base-frame and the laser-frame.

**Amcl** This module is optional. The Adaptive Monte Carlo Localization approach uses a particle filter to track the pose of the robot against a known map (see map_server, the following module). It corrects the robot position,

estimated by the odometry system, moving the odom frame with the respect of the map frame. The less is the error in odometry estimation, the less the amcl module has to correct the position of the odom frame. During the initial phase, in which the robot does not have to navigate autonomously, the amcl module can be missing.

**Map_server** For the amcl module, the map_server is optional because it is used in the autonomous navigation phase. It consists in a node which publish a map previously collected or created.

**Global Costmap and Global Planner** The global costmap carries the information about the obstacles in the map. It is possible to set up an inflation radius which represents a security distance the robot must keep from the walls and other objects. These pieces of information are associated with a cost, and the global planner uses this cost information to find the most efficient path to reach a goal trough the whole map.

**Local Costmap and Local Planner** The local costmap is similar to the global one, but instead of dealing with the whole map, it is localized in a scrolling window around the robot. It is used to modify the global path according to unexpected obstacles, not included in the provided map. The local planner generates a modified path that should not deviate too much from the global one, according to the costs provided by the local costmap.

**Base controller** The Twist messages, i.e., the control signal output from the local planner are sent to the base controller. These messages represent the velocity that the robot should have to follow the generated path. The base controller is the module that interpret these messages and convert them into actual robot movement controlling the wheels' speed.

## 4.2 Sensor fusion and odmetry estimation

The odometry estimation module is basically the most important for localization, mapping and autonomous navigation, since it is involved in all of these processes. In Chapter 3 we discuss about the importance in having a good localization system and the theory behind it; here we discuss about its logical structure.

As mentioned before, we are dealing with a Ra.Ro. version customized for indoor environments, so we cannot count on the GPS, also because the

receiver module is not provided in this version. This implies that the odometry system has to rely on sensors that inevitably accumulates errors.

### 4.2.1 Custom odometry

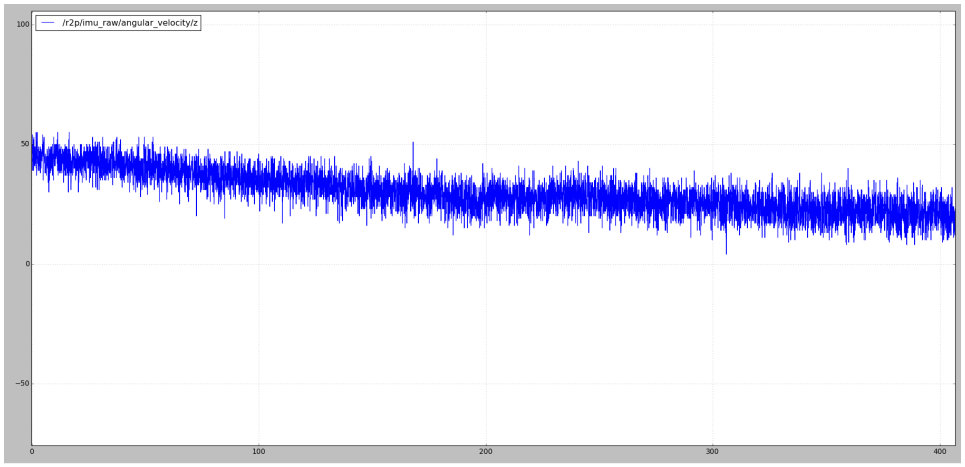
During the initial phase of the project we developed an ad hoc odometry by modifying the already existing Ra.Ro. code producing the odometry from wheels encoders. This modification consists in implementing the Runge-Kutta integration method (3.4) and the exact reconstruction (3.5), instead of the Euler method (3.3), already implemented. Since the exact reconstruction involves a ratio with angle variation $\theta_k$ as denominator, we can use this integration method for $|\theta_k| > 0$. Otherwise the Runge-Kutta integration is used. This is a good solution because the approximation carries an error directly proportional to the angle variation, thus if it used only with very small angles, the subsequent error is small as well. Morover, instead of the $\theta_k$ retrieved from the encoders ROS messages (`/r2p/odom`), we used the ones from the gyroscope (`/r2p/imu`). Here follows the commented code snippet.

```
////////////////////////////////////////////////////////////////////////
// msg.x : is the variation in forward displacement,
// retrieved from wheels' encoders.
// msg.z : is the angle variation,
// retrieved from wheels' encoders.
// imuDeltaZ : is the angle variation,
// retrieved from the gyroscope sensor.
// mSensValues.odom : the struct describing the computed odometry, where
// x and y represent the position into the plane expressed in meters,
// and z the orientation (yaw angle) expressed in radians.
////////////////////////////////////////////////////////////////////////

//Euler method, the most simple, but error sensitive. No more used
//mSensValues.odom.x +=msg.x*cos(msg.z);
//mSensValues.odom.y +=msg.x*sin(msg.z);
//mSensValues.odom.z +=msg.z;

if ( fabs(imuDeltaZ) < 0.0001)// To avoid zero division
{
   //Runge-Kutta method, using angle from gyroscope
   mSensValues.odom.x +=msg.x*cos(mSensValues.odom.z + imuDeltaZ/2);
   mSensValues.odom.y +=msg.x*sin(mSensValues.odom.z + imuDeltaZ/2);
   mSensValues.odom.z +=imuDeltaZ;
}
else
{
```

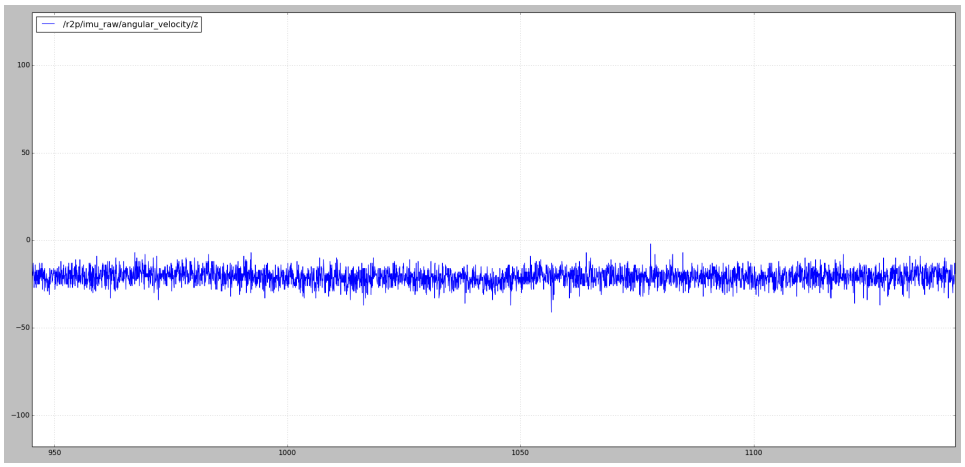
```
    //Precise reconstruction, using angle from gyroscope
    float ratio = msg.x/imuDeltaZ;
    float old_mSensValue_z = mSensValues.odom.z;
    mSensValues.odom.z += imuDeltaZ;
    mSensValues.odom.x += ratio*(sin(mSensValues.odom.z) -
        sin(old_mSensValue_z));
    mSensValues.odom.y -= ratio*(cos(mSensValues.odom.z) -
        cos(old_mSensValue_z));
}
```

The gyroscope sensor is subject to a bias, i.e. the quantitative term describing the difference between the average of measurements made on the same object and its true value. We must take into account this measurement inaccuracy. Moreover our particular sensor bias is not constant, as shown i Figure 4.2, and then we have to frequently update its estimation. A good way to do that has been to implement an observer that correct the gyroscope measurement. In particular our observer checks, using encoders sensor, if the robot stays still, and, if this is true, it updates the gyroscope bias using a low-pass filter, in order to zero it out. Here follows the commented code snippet.

```
// Initialize the angle variation as the difference between the incoming
// angle from gyroscope (msg.z) and the previously saved one
// (mSensValues.imuRaw.z)

double deltaZ = msg.z-mSensValues.imuRaw.z;

//If the encoders yelds that the robot stays still (not moving along the
// x axis, nor rotating around the z axis) and the computed difference is
// not very big (in order to not fit very noisy data, even they are
// filtered in the following line)
if(mSensValues.odomRaw.x == 0.0 && mSensValues.odomRaw.z == 0.0 &&
    fabs(deltaZ) < 0.005)
  {
     //Update the bias estimation using a low-pass filter.
     mGyroZBias = mGyroZBias * 0.9 + deltaZ * 0.1 ;
   }

// Finally update the angle variation with the corrected value
    imuDeltaZ = deltaZ-mGyroZBias;
```




(a) Gyoroscope measurement with the robot stands, from time 0s to time 400s




(b) Gyoroscope measurement with the robot stands, from time 950s to time 1150s

*Figure 4.2: Gyroscope measurement during two different time spans. The time difference is about 550 seconds and we can se how the bias change from an image to another. The represented measurements are raw data from gyroscope. We can see from the images comparison a difference of about 75. Considering that the LSB represents 17.5 mdps, this difference means that exists a bias variation of* $75 \times 17.5 mdps = 1.275 dps$.

### 4.2.2 ROAMFREE module

The most reliable result we had is based on the ROAMFREE framework, introduced in Section 3.3.1. The final set up is based on measurement given by the encoders, the gyroscope and the accelerometer. We also implemented a version with markers as fixed feature position sensor. More details about the set up will be explained in Chapter 5. The main developed ROS nodes are the following:

- `/raroam_test_node`:
  This node builds and manages a factor graph using the ROAMFREE libraries.
  The topic subscribed for measurement retrieving are `/r2p/encoder_l`, `/r2p/encoder_r` and `/r2p/imu_raw`.

  The `/nav_cam/markers` is used as well for markers improvement. The resulting `odom_r` frame is published in a `/tf` topic. This node represents the core of our sensor fusion and subsequent odometry estimation. The parameters about the numbers of Gauss-Newton iterations, the fixed window time and the marginalization time can be modified in the launch file. These parameters heavily influence the quality of the estimation, but are also hardware depending, intended as computational power availability. Another feature of this node is the possibility of publishing the ROAMFREE estimated path as `nav_msgs/Path`, in order to be easily seen though rviz application.

- `/msg_stamper_node`:
  It was necessary to develop this node which simply republishes messages read in r2p's topics and modified adding a header containing, beside the other header's info, a time stamp. This is useful to synchronize the left and right encoder messages and to let ROAMFREE deal with possibly out-of-order messages. The messages are republished in topics with the same name as the original ones, but with the `/stamped` string before them. So `/r2p/imu_raw` became `/stamped/r2p/imu_raw` and so on. The encoders messages are at first matched using the ROS message filter and the republished with the added header in `stamped/r2p/encoders` topic.

#### IMU and encoders setup

One of the most reliable configuration we finally designed is based on the data received from gyroscope, accelerometer in addition to the wheels' encoders one. ROAMFREE implementation does not includes the skid-steering

odometry implementation as logical sensor, but we could take advantage of the Skid-steering to differential drive odometry conversion that we mentioned in 3.2.3. Thus, instead of passing the real distance between the wheels as parameter we passed a value

$$L' = L \cdot \lambda \,, \tag{4.1}$$

where $\lambda$ has been experimentally calculated as

$$\lambda = \frac{\omega_{real}}{\hat{\omega}_{diff}} \,, \tag{4.2}$$

where $\omega_{real}$ is the real angular velocity, and $\hat{\omega}_{diff}$ is the differential drive estimated angular velocity. To retrieve the real angular velocity we used the OptiTrack system installed in the AirLab. It is a motion capture system based on a set of cameras installed around a room, able to filter infrared light, provided with infrared leds rings. Using the leds they can light up a particular marker, made of a rigid frame with reflective marbles, like the one in Figure 4.3. Through the proper OptiTrack software it is possible to retrieve with extreme precision the position and orientation of the marker into the space covered by the cameras. We set a marker on the Ra.Ro. head and we remotely rotated the robot around its vertical axis. We retrieved its orientation, measured both with the OptiTrack and with the wheels odometry, which was set with the real distance between the wheels i.e., as if it was a differential odometry mechanism, instead of a Skid-steering. From these measurements we derived the mean angular velocities $\omega_{real}$ and $\hat{\omega}_{diff}$, using the positions and the delta times, and we computed the ratio, retrieving $\lambda = 1.353491248$. Thus, since the real wheels distance was 0.5m, the resulting equivalent distance was $L' = 0.5 \cdot 1.353491248 = 0.676745624$ which is significantly higher then the real one.

We used as *AngularVelocity* sensor the raw information given by the IMU sensor, as copy of the MEMS memory, properly scaled. The raw values are 16 bit numbers, form -32768 to +32768 representing angular velocity from -500 dps (degree per second) to +500 dps, according to the sensor specifications manual. It means that, we have to make the following conversion:

$$\omega_{deg} = b \cdot 500/32768 \,, \tag{4.3}$$

where $b$ is the raw value and $\omega_{deg}$ the angular velocity expressed in degrees. Then $\omega_{deg}$ needs to be converted in radians, thus, finally we have:

$$\omega_{rad} = b \cdot \gamma \,, \tag{4.4}$$

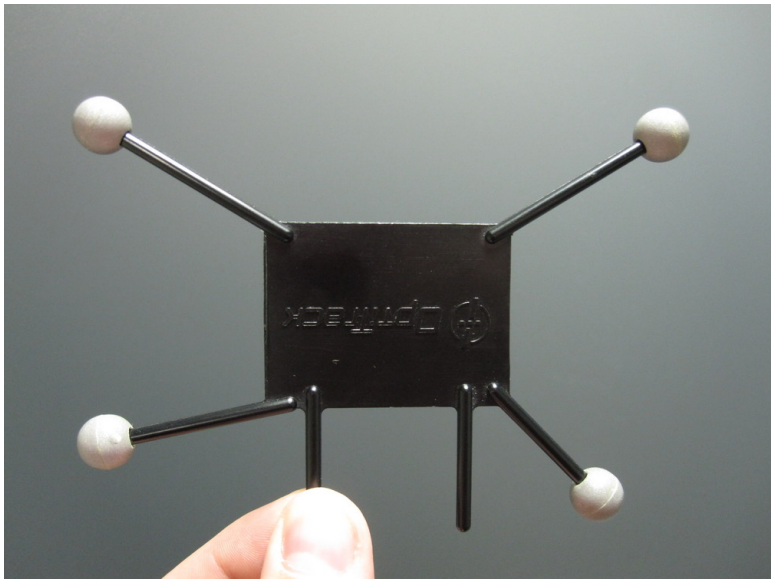


*Figure 4.3: An OptiTrack marker*

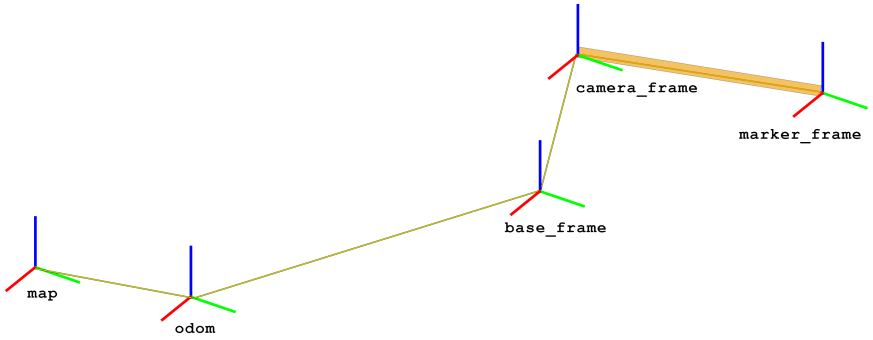
Where $\gamma = \frac{500\pi}{32768 \cdot 180} = 2.66316 \cdot 10^{(} - 4)$. Since errors can happens in sensor production phase, we wanted to estimate this factor experimentally, using the OptiTrack system as well. We obtained a slightly different value, $\gamma' = 3.06844 \cdot 10^{(} - 4)$, which has been used to describe more accurately the real angular velocity.

We should had to retrieve experimentally also the accelerometer proportional factor, but it implies two derivation, because the OptiTrack system retrieve only the position. Since this computation is very error sensitive we used the specification parameters for the *LinearAcceleration* sensor, setting a higher variance.

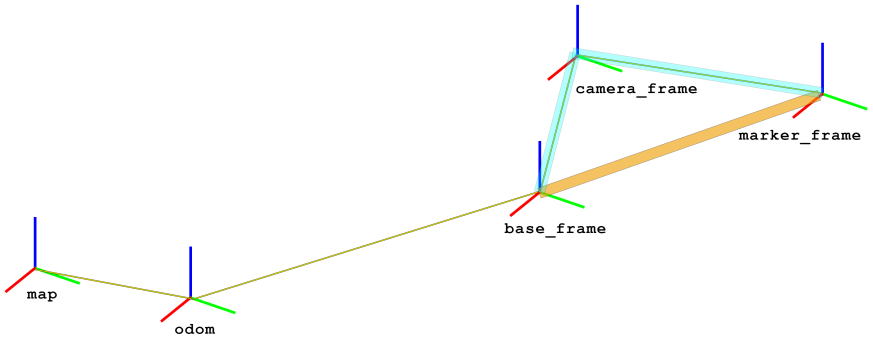
### Using markers for odometry estimation

In addition to the previously mentioned sensors, we introduced the detection of visual markers to have a more accurate odometry estimation. As mentioned before, in Section 2.2, the Ra.Ro. software already provides the aruco marker recognition, which can retrieve the visualized marker identification number and the `tf` transform with the respect to the camera frame.

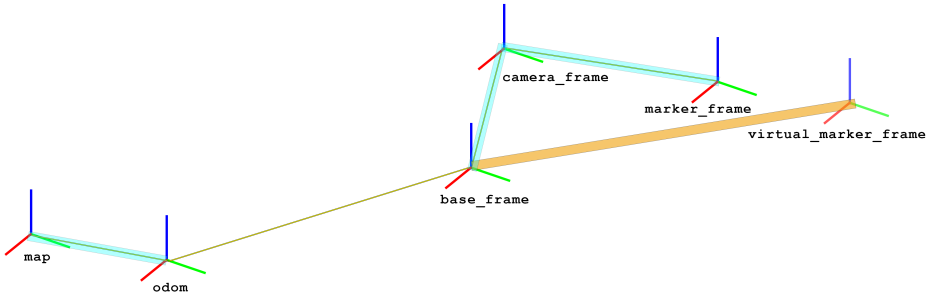
According to the ROAMFREE implementation, each marker must be set up as a single sensor. We could choose to represent these sensors as *FixedFeaturePosition*, which means we use only the retrieved marker position, or *FixedFeaturePose* which includes, in addition, the marker orientation. Since the retrieved orientation is not reliable enough, we decided to represent




(a) The transform required by ROAMFREE




(b) The transform passed to ROAMFREE, setting the base frame as sensor frame




(c) The transform passed to ROAMFREE, during the mapping and autonomous movement phases

*Figure 4.4: Different marker frame measurement passed to ROAMFREE. The ones blue highlated are the tranformation required and composed, the orange ones are the resulting transforamations*

the marker sensors as *FixedFeaturePosition* in order not to introduce errors and having the possibility to use a lower covariance in measurement addition. The measurement required is the observed transformation between the marker and the camera frame.

The ROAMFREE *FixedFeaturePosition* sensor requires the camera frame position, as sensor frame position, and the marker frame position with respect to the camera frame position as measurement, as shown in Figure 4.4(a). The sensor frame position is supposed to be fixed with respect of the base frame, but in our case we are able to rotate the camera according to the robot head module or along its horizontal axis. For this reason we decided to set up the sensor frame as coincident with the base frame and pass as measurement the calculated transformation between the marker frame and the base frame, maintaining the possibility to move the camera, as shown in FIgure 4.4(b).

ROAMFREE requires the absolute marker position as constant parameter, for each *FixedFeaturePosition* sensor. The actual marker positions, in very large buildings like the one we worked into, are not easy to retrieve. To solve this problem we used a solution that introduces a desirable flexibility, indeed. Instead of initialize all the marker set during ROAMFREE launch, we decided to initialize a new marker sensor as soon as the first marker observation, for a specific marker id, is done. It implies that the first marker observation should be done with an odometry error as small as possible, in order not to introduce a wrong placed marker. This is the reason why we introduced the marker sensors as the last feature, in order to have better state estimation where long distance are covered. Indeed, the most desirable case is to see the marker as soon as possible, with the odometry estimation error hopefully small, set it up, and then use the marker to keep the odometry estimation as correct as possible in the long run.

The marker sensors, as here presented, cannot be used if a node moves the odom frame in order to improve the localization, as explained in Section 3.6, because the ROAMFREE framework is an odometry source, so it provides the transform between the odom frame and the base frame, do not considering what happens between the map frame and the odom frame. Thus, what happened if the we use the previous configuration is that, as a marker is viewed, ROAMFREE tries to move the base frame in order to reduce the error between the observed marker and the previously fixed one. In case the odom frame has been moved by another authority, the ROAMFREE correction attempt results as an unpredictable base frame teleportation, which continues to happen as long as a marker measurement is added.

We solve this problem in a way that can seems counterintuitive, but is actually exact and experimentally confirmed. The idea is to pass as the marker observation measurement the composition of the map-odom transform and base-marker transform, as showed in Figure 4.4(c). In this way we

inform ROAMFREE about the transform *behind* the odom frame, and the movement that it applies to the base frame happens to be consistent to the error reduce desired.

## 4.3 SLAM module

At high level the mapping node does not depend on the algorithm used to do the SLAM process. Both `gmapping` and Cartographer, the two mapping systems used here, take as input the messages containing the transforms and the ones containing the laser scans. The localization part of the SLAM node has the aim of virtually move the robot frame in order to let it match the localization estimation, retrieved by means the laser scans data, with the odometry estimation. This movement is done by applying a rigid transformation to the odom frame. The Figure 4.5 can explain better the concept of the relationships between the frames and the nodes that correct them. We can see that the tranform between `odom` and `base-frame` is computed and published by the ROAMFREE node. Basically this is the result of the odometry estimation. AMCL, instead, estimate the `base-frame` position with respect of `map` and correct the actual position moving the `odom` frame with respect of the `map` frame. The AMCL node works between the same nodes as the mapping nodes, as we will explanin in Section 4.4. As output we have the map, periodically updated, and the odom transform, corrected as explained before. Once the mapping phase is done, the final map is saved in a appropriate file, used subsequently in the autonomous navigation phase.

In our definitive architecture we choose to use the `gmapping` framework because we had better results, in particular in case of straight corridors. Indeed, we were able to set up `gmapping` in order to trust more the estimated odometry and the final results are good. We could not reach a so good final map with the Cartographer module, so we present only an example of map genarated by Cartografer in Chapter 5, but the most of our experiment was made using `gmapping` generated maps. The best results we had with the odometry estimation given by the ROAMFREE set up including gyroscope, accelerometer, wheels' encoders and markers recognition. Even if the resulting map is not perfect, for example a slight corridor bend happens, the robot can navigate into the map, only rarely losing the localization.

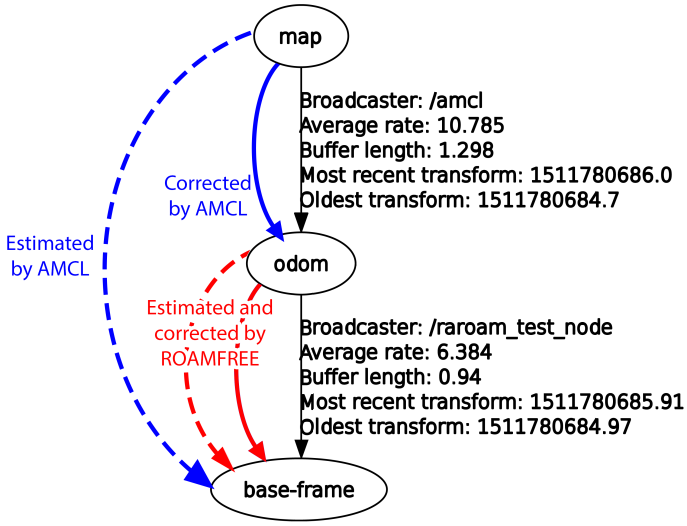



*Figure 4.5: The tree frame representation, with node explanations*

## 4.4 Autonomous navigation module

We decided to work with the most used autonomous navigation module in ROS navigation stack. It is composed by three nodes:`/map_server`, `/amcl` and `/move_base`. With the appropriate set up, the `/amcl` node, combined with the `/move_base` one, is possible to reach a great performance due the high number of configuration parameters available. These modules are highly supported and used by the ROS community, so the work around them was just to setting it up and tune some parameter. We were not interested in having a very fast performance, nor an optimal path planning. We accepted a good solution, being aware that it can be still easily improved working on the configuration parameters.

Here we report some details about the three nodes involved in the autonomous navigation module:

- `/map_server`: It loads a map previously build or drawn and publish it in the `/map` topic.
- `/amcl`: This node wants as input the map, and the robot position in `tf` fashion in order to estimate the position into the map environment. The output is the transform between the map frame and the odometry frame, similarly the `gmapping` approach.
- `/move_base`:
  This node is responsible for reading of a goal point, plan a path be-

tween that goal and the estimate position of the robot, and of sending the actual velocity command for the robot movement. The path is planned using the global and local costmaps, which give the information about the obstacle into the environment, and the global and local path planner which are responsible of plan the desired path to reach the goal point, according to their set up and used algorithms, as explained in Section 4.1

# Chapter 5

# Experiments

*" [Guybrush, tarred and feathered, enters Blondebeard's restaurant.]*
*Captain Blondebeard: ¡Madre de Dios! ¡Es el Pollo Diablo!*
*Guybrush: ¡Sí! ¡He dejado en libertad los prisioneros y ahora vengo por ti!*
*Captain Blondebeard: Well, yer not takin' me without a fight!*
*[Blondebeard bashes Guybrush over the head with a frying pan]"*

The Curse of Monkey Island

In this chapter we illustrate the most significant cases in which we tested our system first. We introduce the general environment setup for the experiments and then we give more details about them.

## 5.1 Setup description

For practical reasons we used the `rosbag` system. It is a ROS package that allows to record ROS data streams and then replay them as if they are happening in real time. It is a good system, widely applied, for tests. We recorder, for our experiments, the following topics:

- `/nav_cam/markers`
- `/odom`
- `/r2p/encoder_l`
- `/r2p/encoder_r`
- `/r2p/imu`
- `/r2p/imu_raw`

- `/r2p/odom`
- `/scan`
- `/scanf`
- `/tf`

to which we added system topics:

- `/rosout`
- `/rosout_agg`
- `/clock`

We based our experiments on the `rosbag` system for several reasons. The first one is for parameters calibration; indeed a careful parameters tuning was needed for a good ROAMFREE setup, and it was important to have a reference dataset in order to allow testing different configurations with the same dataset. Not to causing overfitting, it was important to test the same configuration with different datasets too. Thus, in our case, the datasets were several `rosbag`s. Once the ROAMFREE parameter calibration was considered satisfiable, `rosbag` has been used to compare the different generated odometry, as shown in the following sections.

Another reason to use the `rosbag` system is given from the possibility of replay a bag slower than normal. It is a useful feature because it allows to run code that can require to much effort for real time usage. For instance, in the mapping phase, the ROAMFREE system, with specific configurations, can be computationally heavy, and a slower replay is appreciate. This can be acceptable even in a production phase, since the robot can easily do the exploration for mapping purposes, recording a `rosbag`, and then it just need to wait some time more for the actual map building in the replay phase.

The experiments focusing on the mapping phase, reproducing the bags at the 70% of the real time speed, however an adequate hardware, like the one installed into the robot, can satisfy the required computing power, without affect the final map result. Finally a practical reason to use the `rosbag` system can be mentioned, i.e. because of it was not very handy to let run the robot into the laboratory environment tens of times, both for other people trouble, made or been made, and for save the robot from early wear.

The main `tf` frames involved in all the experiments are: `map`, `odom`, `base-frame`, `base-frame-custom`. In particular the `base-frame` is the one

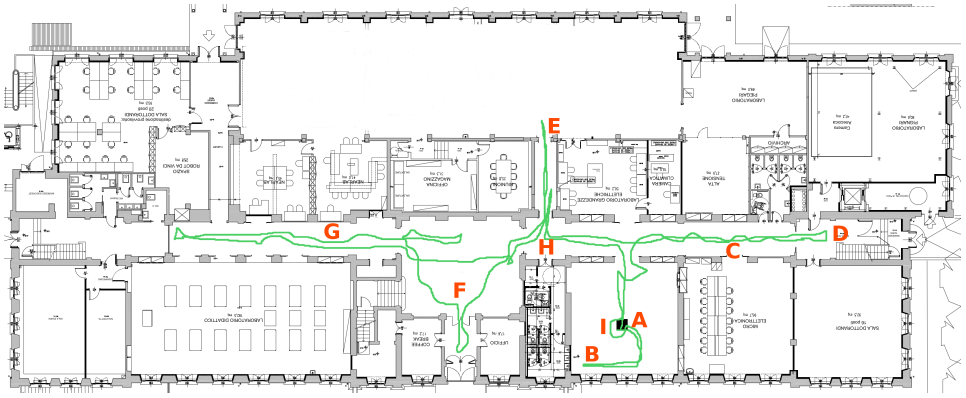



*Figure 5.1: An approximate hand drown ground truth of the path made by the robot*

involved every time a ROAMFREE odometry is published, `base-frame-custom` when the odometry is estimated by the custom odometry system.

## 5.2 Odometry experiments

In this section we focus on the experiments involving only the odometry estimation. In particular we compare the three main configurations we think are the most significant. We recorded a `rosbag` in which we drove the robot in manual configuration. We can summarize the path followed in these steps:

A. Start at the black square into the AirLab.

B. Have a round into the room and then go out through the door.

C. Go right until the end of the corridor is reached.

D. Turn around an go back until the vertical corridor is reached.

E. Follow the corridor and back again.

F. Continue to the left corridor moving through big room and go inside the small room connected to the big one.

G. Continue to the left corridor until the end is reached and then go back following an almost straight line.

H. Go again through the vertical corridor for a little and then come back to the AirLab.

I. Finish the ride in the same point we started.

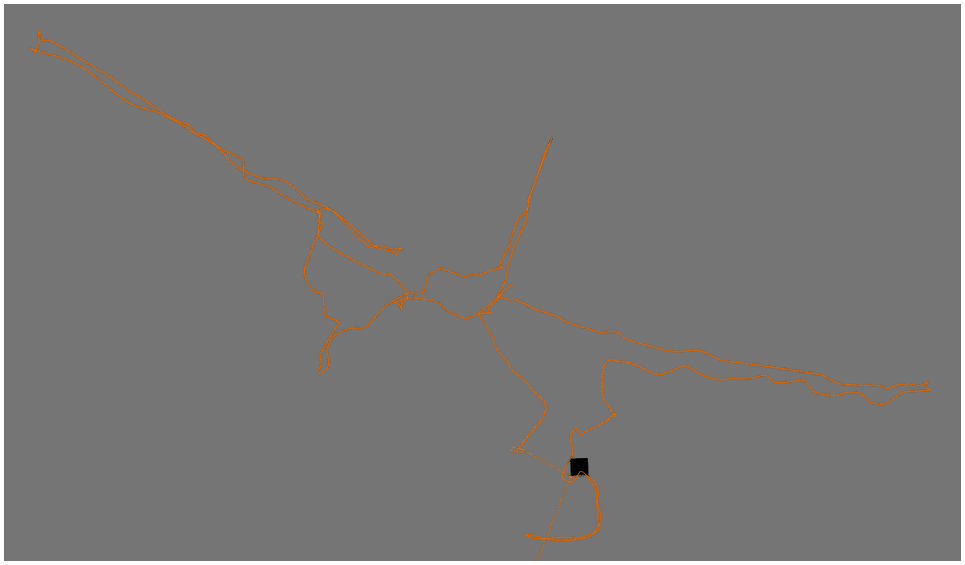


*Figure 5.2: The resulting path generated with the custom odometry*

An idea of how the followed path should seems is in Figure 5.1. We ran the same `rosbag`, each time setting a different odometry estimation method: custom odometry, ROAMFREE odometry with IMU and encoders, ROAMFREE odometry with IMU, encoders and markers. We compare the resulting paths with a real floor plan of the building. Because we are focusing only on the odometry part, the map frame will be the same as the odom frame. We want to point out that none of these odometry estimation uses the LIDAR sensor, which is usually the best sensor to rely on.

### 5.2.1 Custom odometry

For this experiment we used only the custom odometry; the one built integrating the gyroscope and the encoders data, described in Section 4.2.1. We can see the resulting path in Figure 5.2 and we can compare it with the supposed ground truth in Figure 5.1. We can notice that the initial part, inside the AirLab is good enough, and the path that continues along the corridors, from right to left, is almost straight. During the return part, from left to right, we can see that, when we drove the robot inside the vertical corridor again, the path are very close to match. Finally, the end point diverges form the initial one for about 3 meters, according to the misuration retrieved from `rviz`, out of the more than 160 meters traveled, according to the measurement retrieved from the reference floor plan. As reference, the black square in Figure 5.2 is 1m×1m wide.

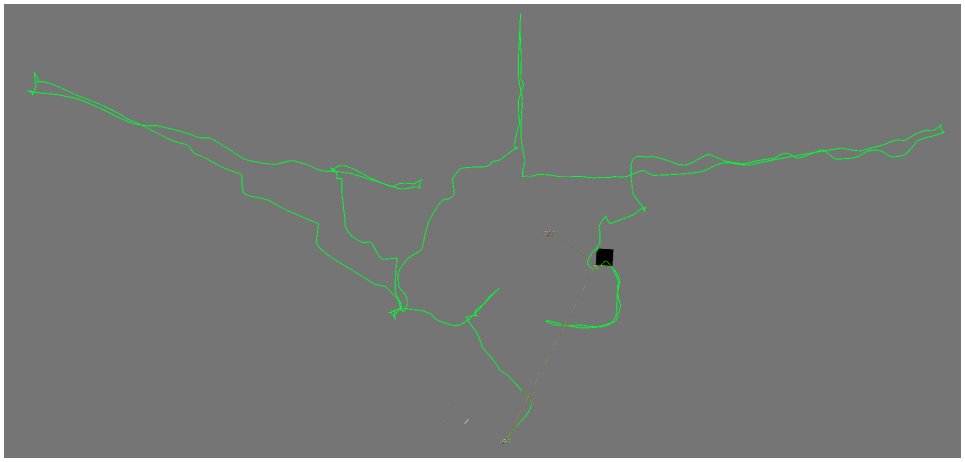


*Figure 5.3: The resulting path generated with the ROAMFREE odometry, without markers*

Therefore, we can say that the custom odometry diverge slowly, but constantly. This kind of odometry has the limitation that is not possible to improve, due the way in which it is implemented, i.e., without any kind of modularity. However, since the error grows slowly, if needed, a periodic reinitialization can correct the error occurred.

### 5.2.2 ROAMFREE odometry

For the second experiment we set the ROAMFREE parameters as presented in Table 5.1. We set the `DifferentialDriveOdometer` as master sensor, setting the $6 \times 6$ covariance matrix with 0.01 diagonal values. This implies a quite high importance for the odometry retrieved from encoders. For the `AngularVelocity` sensor, we decided to fix the angular velocity measurement around the `x` and `y` axis to 0.0, being a differential drive mechanism, in order to not introduce possible errors in estimation. For this reason we set very low variance for these two component, and 0.02 for the `z` component, i.e. the angular velocity around the vertical axis, which is the only one we are interested on. About the `LinearAccelerometer` sensor we decided to set the `z` component of the measurements as a fixed value, equal to -9.81, which is the $g$ acceleration. Thus, we set a low variance for this component and a high variance for `x` and `y` components. We decided to use a high variance for these component because the accelerometer sensor is not very accurate and difficult to manage, due the two integration needed to retrieve position information.

We can see that for more then half of the traveled path the result is good,

better then the one with the custom odometry. After the round into the big room the path bends, probably due to a bad gyroscope bias estimation, and this persists for all the following part of the ride, finishing far from the initial point. We can notice that also during the last part of the ride, the path remains straight and if we try to rotate this part -around the end of the corridor on the left- we can see that the final path almost matches the real one, taking into account the small error accumulated during the previous part of the path.

It is necessary to point out that replaying the same bag we can occur in slightly different results. This can happen because of the heavy mathematical computations done by the framework, which sometimes need some approximation which can became significant after various multiplications.

Similarly to the custom odmetry we have an odometry that works well for the initial part and then starts to drift. We can notice that in this case the error does not grow constantly, and it can happens that a bad angle estimation leads the odometry to fail. For this reason a good solution could be include the marker recognition as in the next experiment.

*Table 5.1: ROAMFREE configuration parameters*

| Logical Sensor (sensor) | Measurement covariance |
|---|---|
| `DifferentialDriveOdometer` (Encoders) | $\begin{bmatrix} 0.01 & 0.0 & \cdots & 0.0 \\ 0.0 & 0.01 & \cdots & 0.0 \\ \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & \cdots & 0.01 \end{bmatrix}$ |
| `AngularVelocity` (Gyroscope) | $\begin{bmatrix} 0.0001 & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0 \\ 0.0 & 0.0 & 0.02 \end{bmatrix}$ |
| `LinearAcceleration` (Accelerometer) | $\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0001 \end{bmatrix}$ |

### 5.2.3 ROAMFREE with markers odometry

The ROAMFREE with markers odometry significantly improves the previous odometry. We set up the ROAMFREE environment as the previous one, except for the addition of marker sensors as presented in Table 5.2. We

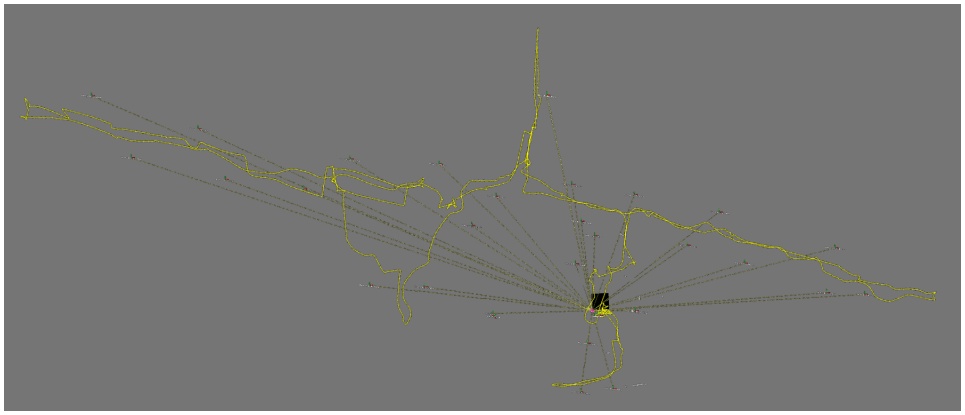


*Figure 5.4: The resulting path generated with ROAMFREE odometry and markers*

point out that the marker position was unknown at the beginning of the ride. Each marker was added as new sensor as soon it has been seen for the first time. This implies that a good odometry, without markers, is necessary to initialize markers sensors properly. Indeed, markers are landmarks which are memorized the first time they are seen, then, the following times, their recognition adjust the odometry according to the position in which the marker is seen again.

Since ROMAFREE basis is a factor graph, the marker measurements improve and refine also the nodes before the incoming measurement. This gives a better estimate of the gyroscope bias and improves the odometry estimation itself. Markers add to the system the missing fixed point reference that we needed because of GPS unavailability, leading a very good odometry estimation, finally.

It is important to point out that we used markers as fixed points, but, due the ROAMFREE modularity, it can be possible to substitute markers with other kind of fixed features retrieved by cameras or other sensors. In the Figure 5.4 we can also see the generated markers constellation.

*Table 5.2: Markers in ROAMFREE configuration parameters*

| Logical Sensor (sensor) | Measurement covariance |
|---|---|
| `FixedFeaturePosition` (Markers) | $\begin{bmatrix} 0.0001 & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0 \\ 0.0 & 0.0 & 0.0001 \end{bmatrix}$ |

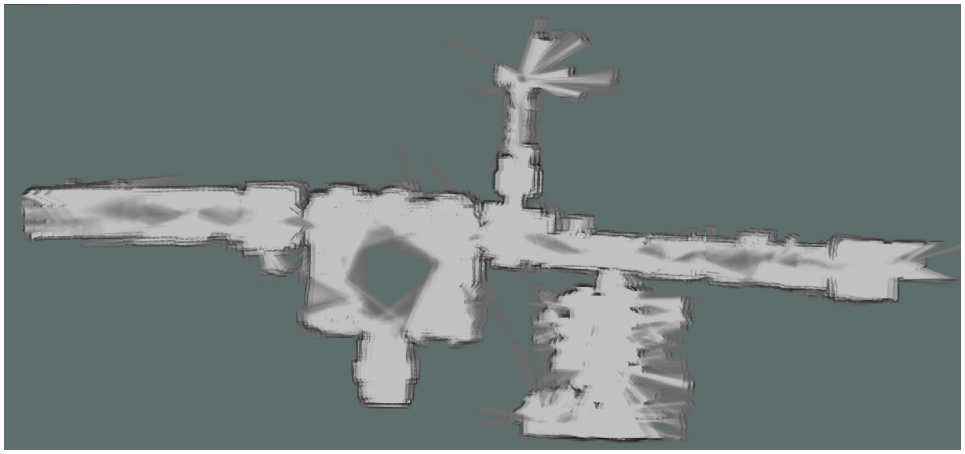


*Figure 5.5: Map generated using Cartographer. The long corridors result to be too short and the borders does not match properly.*

Markers addition as ROAMFREE sensor needs to be carefully weighted. We used a very low covariance because working in a controlled environment. Problems can occur if a marker is moved from its position to another one, far from it, after the first recognition. If this risk exists, it should be a good practice to increase the covariance in measurement adding.

## 5.3 Mapping experiments

In the final map experiments we decided to use only the `gmapping` tool, because we could not properly set Goolge's Cartographer in order to trust more the odometry system, instead of rely only on laser measurement, causing a vary bad map creation in the long corridors we had to deal with. As show in in Figure 5.5 both the long corridors results to be much more shorter than real ones. Moreover the edges representing the obstacles happens to not properly match. Both the issues make the Cartographer map difficult to be properly navigated. We could set up `gmapping` for this purpose and the most significant parameters are reported in 5.3. We decided to use a different `rosbag` for the mapping phase in order to not generate overfitting.

### 5.3.1 Mapping with custom odometry

Using the custom odometry we retrieve a good map, slightly bended along the long corridors, which is a typical error in mapping. The length of corridors is very similar to the real ones, although we could expect another typical problem in mapping long corridors: their shortening. The laser scans and

*Figure 5.6: The resulting map generated with the custom odometry*

the `gmapping` algorithms can compensate the error caused in odometry estimation, seen before, creating a well navigable map. The resulting map can be seen in Figure 5.6

For this set up we completely deactivated the ROAMFREE module and we set a static transform between the `base-frame-custom` and the `base-frame` in order to not be needed to change the `gmapping` reference frame.

### 5.3.2 Mapping with ROAMFREE odometry

The resulting map, using the ROAMFREE odometry, without markers, is slightly better then the custom odometry one, but very similar. In this case we left both the custom odometry and ROAMFREE odometry running because of the poor modularity into the robot system, but we set the

*Table 5.3: `gmapping` parameters*

| Parameter | Default value | Used Value | Description |
|---|---|---|---|
| srr | 0.1 | 0.01 | Odometry error in translation as a function of translation (rho/rho) |
| srt | 0.2 | 0.02 | Odometry error in translation as a function of rotation (rho/theta) |
| str | 0.1 | 0.01 | Odometry error in rotation as a function of translation (rho/rho) |
| stt | 0.2 | 0.02 | Odometry error in rotation as a function of rotation (rho/rho) |

*Figure 5.7: The resulting map generated with the ROAMFREE odometry without markers*

`gmapping` system to run with the `base-frame` published by ROAMFREE. The resutling map is represented in Figure 5.7.

### 5.3.3 Mapping with ROAMFREE odometry and markers

The map generated with the ROAMFREE odometry with markers is in a way better, because the corridors results less bended, but on the other hand they happens to be a little bit shorter than the real ones possibly, causing poor scan matching were the corridor finishes into the big room. The TF tree in this case includes the saved markers frames attached to the map frame. The resulting map can be seen in Figure 5.8

## 5.4 Navigation experiments

We also tested the navigation system selecting the last map generated with ROAMFREE which includes markers odometry, and switching among the different estimation odometry systems in order to compare them in navigation phase. Due to the good map generated, in every case the error caused by the odometry estimation can be well corrected by AMCL, which improves the localization into the map using the laser scans. For this reason we can conclude that all the odometry system, together with the `move_base` module, can let the robot navigate autonomously inside the generated map, reaching a given goal without loosing its localization.

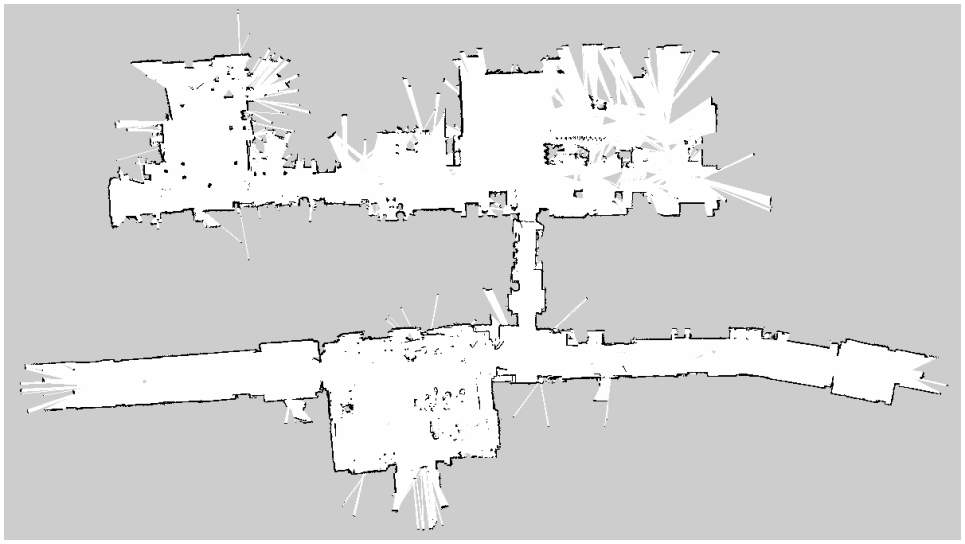


*Figure 5.8: The resulting map generated with the ROAMFREE odometry with markers*

The main problems inccurred during the navigation were caused by obstacles invisible to the laser scanner because of the height lower then the laser position, causing the robot crashing into them and getting stuck for a while. In some situation, the recovery system, which consists basically in turning the robot around its vertical axis looking for laser features to match, was able to relocate the robot, no matter which odometry estimation system was involved.

Another problem occurred probably because of the oldness of the platform. Some of the wheels are no more perfectly aligned. This fact implies that the trajectory must be continuously corrected. If the robot is manually driven, this correction is done by the human almost subconsciously, but if the robot is driven by the local planner it is more complex and the result is a *swinging* trajectory.

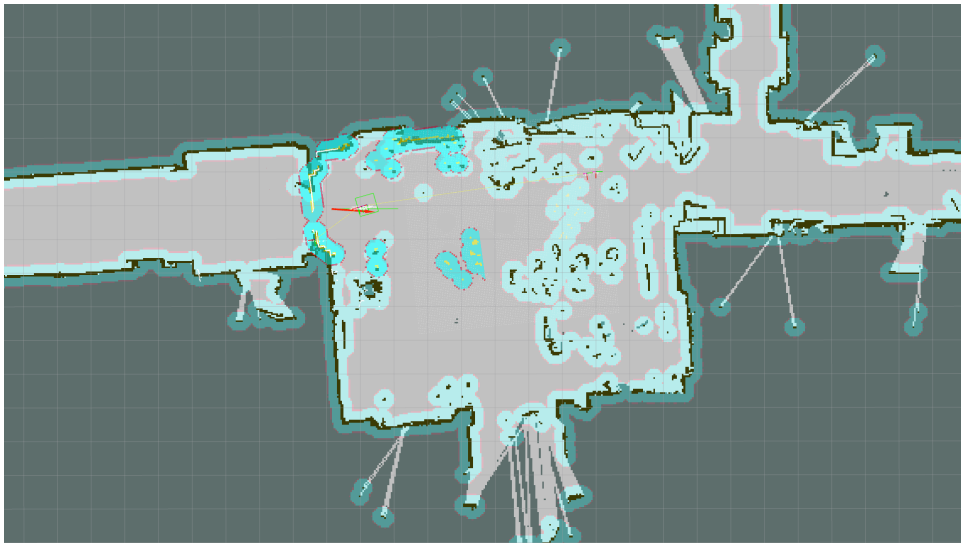


*Figure 5.9: The `rviz` visualization of the local and global costmaps*

# Chapter 6

# Conclusion and Future Work

*"Guybrush: At least I've learnt something from all of this.*
*Elaine: What's that?*
*Guybrush: Never pay more than 20 bucks for a computer game.*
*Elaine: A what?*
*Guybrush: I don't know. I have no idea why I said that."*

The Secret of Monkey Island

The main purpose of this thesis was to build an autonomous navigation system for the provided Ra.Ro. platform, reliable at least as much as the built-in semi-autonomous navigation, but more powerful, able to deal with obstacles, flexible and non environment invasive. The focus was on the sensor fusion in order to retrieve a good odometry, which is involved in all the pieces of the navigation stack, both for the map building, and the autonomous navigation system itself.

Both the ROAMFREE solutions, in particular the one with the marker, can be considered a good solution for the sensor fusion problem, moreover the custom odometry, even if is not a so flexible solution, is even more reliable in some situations for our specific case. The final result, taking into account the physical platform limitation, are satisfying from our point of view and the NuZoo point of view, as well.

We had to deal with the most common problem in indoor environment: the absence of a GPS measurements. The ROAMFREE library was mostly used in outdoor environment and, when used in indoor ones, the markers were heavily used. Another solution in indoor environment was to use a pair of laser scanner, covering the whole range around the vehicle, and set as ROAMFREE input the laser odometry generated. It was impossible in our case because the poor laser scanner placement.

A ROAMFREE extension can be appreciated, in particular a future work can include the development of a SLAM system integrated in the ROAMFREE framework, in order to include the use of the laser scanner as main sensor, through the scan matching process. In the meanwhile, the creation of the map should be an easy task, once the previous part is done. It will avoid strange tricks, as the one we had to use to properly take advantages of the marker as sensor. Moreover a system that integrates properly the sensor fusion and a map generation using a factor graph can potentially be a good solution for both the odometry estimation and SLAM problems.

# Appendix A

# About ROS and the TF library

ROS stands for Robot Operating System and the official ROS introduction web page introduce the system with these words: "ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers." [33]. Thus, ROS is not a real operating system, but it but instead works alongside a traditional operating system. It is designed and optimized to work with Ubuntu, but beta versions for other operating systems exist, e.g. Windows, OSX. ROS relies on a good and active community which makes it the standard *de facto* for robotics in research fields, but also in hobby and professional fields.

One of the main ROS advantages is its modularity, and the modules are called packages. Packages provide functionality such as controlling movement of the robot, generating odometry information, reading and processing sensor data from e.g. a Kinect, camera or laser scanner sensor, keeping track of a robots joint configuration, etc. Packages can also provide more high level functionality such as SLAM implementations, object recognition, 3D simulation, compatibility layers to enable the use projects like Point Cloud Library (PCL) and Open Computer Vision (OpenCV). A common set of packages is set up in the full-desktop installation, which allow to use all the basic packages and includes other useful packages needed, for example, for data visualization, simulation and so on. ROS can be run in a distributed way on multiple computers and the communication among them

is done over TCP.

## A.1 ROS Filesystem Level

ROS uses various concepts at local filesystem level, including:

**Packages** : At computer local filesystem, ROS organizes functionalities in packages. They contains files that together give the package functionality, such as executable (*/nodes*), source code, cmake files, ROS message type definitions, ROS service type definitions, roslaunch files, configuration files and so on. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package.

**Metapackages** : Metapackages are specialized Packages which only serve to represent a group of related other packages. Most commonly metapackages are used as a backwards compatible place holder for converted rosbuild Stacks.

**Package Manifests** : Manifests (`package.xml`) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.

**Repositories** : A collection of packages which share a common Version Control System (VCS). Packages which share a VCS share the same version and can be released together using the catkin release automation tool bloom. Often these repositories will map to converted rosbuild Stacks. Repositories can also contain only one package.

**Message (msg) types** : Message descriptions define the data structures for messages sent in ROS.

**Service (srv) types** : Service descriptions define the request and response data structures for services in ROS.

## A.2 ROS Computation Graph Level

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of

ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

**Nodes** : Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as roscpp or rospy.

**Master** : The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

**Parameter Server** : The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.

**Messages** : Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

**Topics** : Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption.

**Services** : The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request

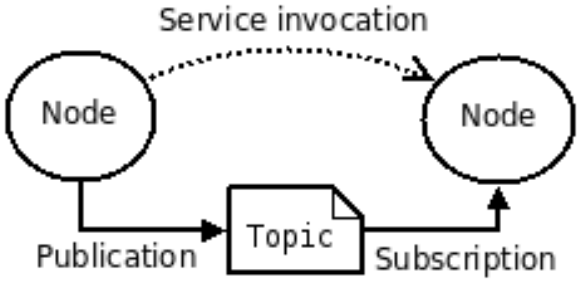



*Figure A.1: The ROS Computations Graph*

message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

**Bags** : Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

The image A.1 gives a rough idea about the connection among nodes, topics and services.

## A.3 The TF library

`tf` is one of the most used and useful package in the ROS system. It is responsible for keeping track of multiple coordinate frames over time. `tf` maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time [34].

A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. `tf` keeps track of all these frames over time, and allows you to ask questions like:

- Where was the head frame relative to the world frame, 5 seconds ago?
- What is the pose of the object in my gripper relative to my base?
- What is the current pose of the base frame in the map frame?

`tf` can operate in a distributed system. This means all the information about the coordinate frames of a robot is available to all ROS components

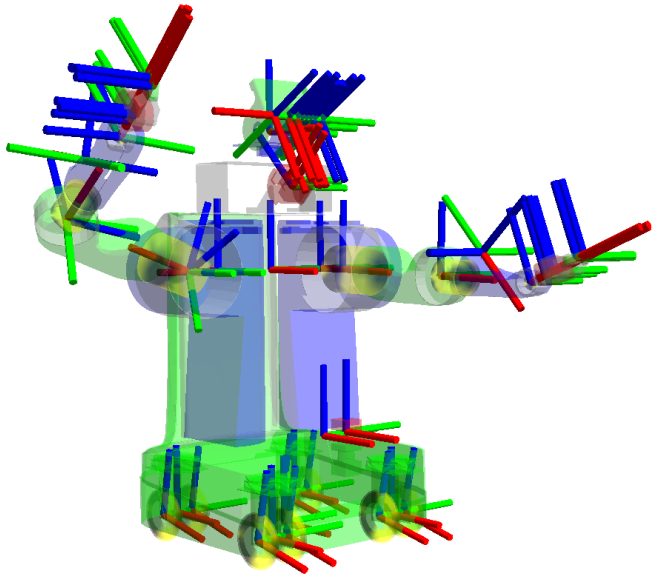


*Figure A.2: A representation of a robot with several frames.*

on any computer in the system. There is no central server of transform information.

There are essentially two tasks that any user would use `tf` for, listening for transforms and broadcasting transforms.

- **Listening for transforms** - Receive and buffer all coordinate frames that are broadcasted in the system, and query for specific transforms between frames.
- **Broadcasting transforms** - Send out the relative pose of coordinate frames to the rest of the system. A system can have many broadcasters that each provide information about a different part of the robot.

# Appendix B

# Sensors specifications

In this section we will show a detailed list of sensors specifications built into the Ra.Ro. platform.

## B.1 LSM303D
## Ultra compact high performance e-compass: 3D accelerometer and 3D magnetometer module

This module is built into the r2p board. The r2p board receives the data form this module and made some low level elaboration. The data computed are sent to the NUC connected to them. The LSM303D module includes a 3D accelerometer and 3D magnetometer. We will show the table about both the sensors, even if in our project the magnetometer was not eventually used.

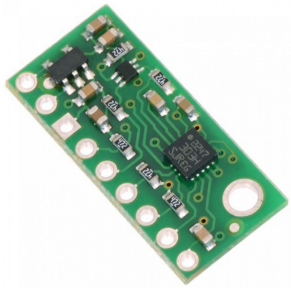


*Figure B.1: LSM303D module*

| **Parameter** | **Test conditions** | **Min.** | **Typ.** | **Max.** | **Unit** |
|---|---|---|---|---|---|
| Linear acceleration measurement range | | | ±2 | | $g$ |
| Magnetic measurement range | | | ±2 | | gauss |
| Linear acceleration sensitivity | Linear acceleration FS = ±2$g$ | | 0.061 | | m$g$/LSB |
| Magnetic sensitivity | Magnetic FS = ±2 gauss | | 0.080 | | mgauss/LSB |
| Linear acceleration sensitivity change vs. temperature | | | ±0.01 | | %/°C |
| Magnetic sensitivity change vs. temperature | | | ±0.05 | | %/°C |
| Linear acceleration typical zero-$g$ level offset accuracy | | | ±60 | | m$g$ |
| Linear acceleration zero-$g$ level change vs. temperature | Max delta from 25°C | | ±0.5 | | m$g$/°C |
| Linear acceleration noise density | Linear acceleration FS = 2$g$; ODR = 100 Hz | | 150 | | ug/($\sqrt{Hz}$) |
| Magnetic acceleration noise density | Magnetic FS = 2 gauss; LR setting CTRL5 (M_RES[1,0]) = 00b | | 5 | | mgauss/RMS |
| Magnetic cross-axis sensitivity | Cross field = 0.5 gauss Applied = ±3 gauss | | ±1 | | % FS/gauss |
| Maximum exposed field | No permanent effect on sensor performance | | | 10000 | gauss |
| Magnetic disturbance field | Sensitivity starts to degrade. Automatic S/R pule restores the sensitivity | | | 20 | gauss |
| Operating temperature range | | -40 | | +85 | °$C$ |

## B.2 L3GD20H MEMS motion sensor: three-axis digital output gyroscope

This module is built in the r2p board as well. It is the gyroscope sensor, able to provide angular velocity information.

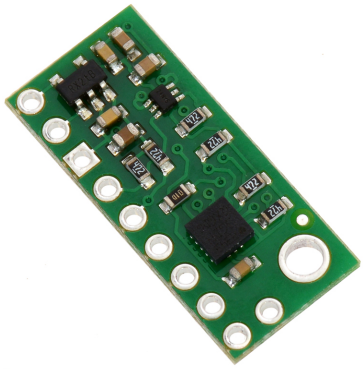


*Figure B.2: L3GD20H module*

| Parameter | Test conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|
| Measurement range | User selectable | | ±245<br>±500<br>±2000 | | dps |
| Sensitivity | | | 8.75<br>17.50<br>70.00 | | mdps/digit |
| Sensitivity change vs. temperature | From -40°C to +85°C<br>Delta form T = 25° | | ±2 | | % |
| Digital Zero-rate level | FS = 2000 dps | | ±25 | | dps |
| Zero-rate level change vs. temperature | FS = 2000 dps | | ±0.04 | | dps/°C |
| Non linearity | Best fit straight line | | 0.2 | | %FS |
| Rate noise density | BW = 50 Hz | | 0.011 | | dps/ ($\sqrt{Hz}$) |
| Digital output data rate | | | 11.9/<br>23.7/<br>47.3/<br>94.7<br>/189.4<br>/378.8<br>/757.6 | | Hz |
| Magnetic disturbance field | Sensitivity starts to degrade. Automatic S/R pule restores the sensitivity | | | 20 | gauss |
| Operating temperature range | | -40 | | +85 | $°C$ |

## B.3 Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder

The Hokuyo URG-04LX-UG01 is a Laser Scanner. It is used in our project for the scan matching by `gmapping`, for map generation, and by AMCL for the localization and autonomous navigation.


*Figure B.3: Hokuyo URG-04LX-UG01*

| **Parameter** | **Value** |
|---|---|
| Light source | Semiconductor laser diode ($\lambda = 785$nm),<br>Laser safety Class 1<br>Laser power: 0.8mW or less (Class 1 compliant by scanning) |
| Power source | 5V DC ±5% (Supplied by USB bus power) |
| Current consumption | 500mA or less (Rush current 800mA) |
| Detection distance and standard object | Accuracy: 60-4.095mm (white paper 70mm × 70 mm or bigger)<br>Detectable range: 20-5.600mm |
| Accuracy | 0.06-1m: ±30mm , 1-4m: 3% of the detected distance |
| Resolution | 1mm |
| Scan angle | 240° |
| Angular resolution | Approx. 0.36° (360°/1024) |
| Scan Time | 100msec/scan |
| Interface | USB Version 2.0 FS mode (12Mbps) SCIP2.0 |
| Ambient (Temperature/Humidity) | -10 ∼ 50°C / 85% or less (without dew and frost) |
| Preservation temperature | -25 ∼ 75°C |
| Ambient Light Resistance | 10000Lx or less (Sunlight) |
| Vibration Resistance | Double amplitude 1.5mm 10 ∼ 55Hz, 2 hours each in X, Y and Z direction, and 98m/s$^2$ 55Hz ∼ 150Hz in 2 minutes sweep, 1 hour each in X, Y and Z direction |
| Impact Resistance | 196 m/s$^2$, 10 times each in X, Y and Z direction |
| Protective Structure | Optics : IP64<br>Case : IP40 |
| Insulation Resistance | 10MΩ for DC 500Vmegger |
| Weight | Approx. 160g |
| Case | Polycarbonate |
| External dimension (W×D×H) | 50×50×70mm |

# Bibliography


[1] Nuzoo website - ra.ro. Accessed: Dec 2016.

[2] Novalab website. Accessed: Jan 2017.

[3] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics (intelligent robotics and autonomous agents). 2005.

[4] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[5] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[6] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, 1993.

[7] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[8] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010.

[9] Timothy A Davis. *Direct methods for sparse linear systems*. SIAM, 2006.

[10] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[11] Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman, and Frank Dellaert. Eliminating conditionally independent sets in factor graphs: A

unifying perspective based on smart factors. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4290–4297. IEEE, 2014.

[12] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[13] Turtlebot robot. Accessed: Oct 2017.

[14] Robert M Harlan, David B Levine, and Shelley McClarigan. The khepera robot and the krobot class: a platform for introducing robotics in the undergraduate curriculum. *ACM SIGCSE Bulletin*, 33(1):105–109, 2001.

[15] Emmanuel Lomba and Mário Alves. On the hardware and software architecture of the robuter mobile platform: a hands-on approach. Technical report, CISTER-Research Centre in Realtime and Embedded Computing Systems, 2005.

[16] Luca Caracciolo, Alessandro De Luca, and Stefano Iannitti. Trajectory tracking control of a four-wheel differentially driven mobile robot. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 4, pages 2632–2638. IEEE, 1999.

[17] Jingang Yi, Dezhen Song, Junjie Zhang, and Zane Goodwin. Adaptive trajectory tracking control of skid-steered mobile robots. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2605–2610. IEEE, 2007.

[18] Jingang Yi, Junjie Zhang, Dezhen Song, and Suhada Jayasuriya. Imu-based localization and slip estimation for skid-steered mobile robots. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2845–2850. IEEE, 2007.

[19] Krzysztof Kozłowski and Dariusz Pazderski. Modeling and control of a 4-wheel skid-steering mobile robot. *International journal of applied mathematics and computer science*, 14:477–496, 2004.

[20] Jorge L Martínez, Anthony Mandow, Jesús Morales, Salvador Pedraza, and Alfonso García-Cerezo. Approximating kinematics for tracked mobile robots. *The International Journal of Robotics Research*, 24(10):867–878, 2005.

[21] Tianmiao Wang, Yao Wu, Jianhong Liang, Chenhao Han, Jiao Chen, and Qiteng Zhao. Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. *Sensors*, 15(5):9681–9702, 2015.

[22] Jingang Yi, Hongpeng Wang, Junjie Zhang, Dezhen Song, Suhada Jayasuriya, and Jingtai Liu. Kinematic modeling and analysis of skid-steered mobile robots with applications to low-cost inertial-measurement-unit-based motion estimation. *IEEE transactions on robotics*, 25(5):1087–1097, 2009.

[23] Anthony Mandow, Jorge L Martinez, Jesús Morales, José L Blanco, Alfonso Garcia-Cerezo, and Javier Gonzalez. Experimental kinematics for wheeled skid-steer mobile robots. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1222–1227. IEEE, 2007.

[24] Davide Antonio Cucci and Matteo Matteucci. A flexible framework for mobile robot pose estimation and multi-sensor self-calibration. In *ICINCO (2)*, pages 361–368, 2013.

[25] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[26] Ingemar J Cox. Blanche: Position estimation for an autonomous robot vehicle. In *Autonomous robot vehicles*, pages 221–228. Springer, 1990.

[27] Sean P Engelson and Drew V McDermott. Error correction in mobile robot map learning. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2555–2560. IEEE, 1992.

[28] N.L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous univariate distributions*. Number v. 2 in Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley & Sons, 1995.

[29] J.A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury advanced series. Thompson/Brooks/Cole, 2007.

[30] Amcl documentation. Accessed: Jan 2017.

[31] Ros rep 105. Accessed: Jan 2017.

[32] Ecef wikipedia. Accessed: Oct 2017.

[33] Ros wiki. Accessed: Oct 2017.

[34] Tf wikiros. Accessed: Nov 2016.