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

A qualcuno...

Abstract

One of the most useful purpose in self driving autonomous robots is to substitute for human activity in dangerous and heavy jobs. The Ra.Ro platform, developed by NuZoo, is designed to work as a security robot, able to patrol, detect and eventually send alarm to a centralized station. This platform is often required to be adapted to different purposes, but most of them require an autonomous navigation. The aim of this thesis is to propose a mapping and localization system to avoid the well-known drifting problem. In particular we are dealing with the indoor environment, which is challenging because is not available any fixed point reliable sensor such as GPS. 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.

The final results, tested in the real world, are quite satisfying at the end, but can be further improved.

Sommario

Uno dei principali scopi di robot a guida autonoma è quello di sostituire l'uomo nelle attività più pericolose a nei lavori più pesanti. La piattaforma Ra.Ro., sviluppata da NuZoo, è progettata per lavorare come robot di sicurezza, capace di pattugliare, individuare e infine lanciare un allarme ad una stazione centralizzata. Spesso è stato richiesto che questa piattaforma venga adattata per diversi scopi, ma la maggior parte di essi richiedono una navigazione autonoma. L'obbiettivo di questa tesi è di proporre un sistema di mappatura e localizzazione che ovvi al noto problema del *drifting.* In particolare ci occupiamo di un ambiente interno, il quale è difficoltoso a causa della mancanza di un sensore, come il GPS, che ci dia informazioni affidabili su dei punti fissi. Il sistema di odometria fornito dagli *encoder* delle ruote non è abbastanza preciso e molto sensibile agli errori, per questo è molto importante fondere le infomazioni ricevute da molteplici sensori quali una IMU, un LIDAR e una videocamera utilizzata per riconoscere marker specifici.

I risultati, testati in un ambiente reale, sono stati soddisfacenti alla fine, ma possono essere ancora migliorati.

Ringraziamenti

Ringrazio ....

Contents

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 so common. Certainly 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 actual 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, certainly the ability to move autonomously in the environment is very appreciated, regardless of the high-level purpose of the robot. 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 potential problems are easy to identify. First of all, in some locations 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 ill-intentioned person to cover, delete or detach them, getting the robot lost in seconds.

1.1 Thesis contribution

The Ra.Ro. platform had already been 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.

This thesis proposes a multi-sensor navigation system based on the ROAM-FREE framework, a system that provides a multi-sensor fusion tools to improve the odometry estimation using the information provided by the different sensors. We decided to use the wheel encoders, gyroscope, accelerometer and, for a better result, markers as landmarks.

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 software environment. In particular we describe the ROS topics used by the robot and the built in navigation systems in the following subsections. Moreover briefly write about some Ra.Ro. purposes.

- In chapter 3 we describe the knowledge needed to deeply understand in what our project consist and some state of the art examples. In particular section 3.1 we introduce 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 frameworks we applied in this thesis. The following section 3.4 is about SLAM, simultaneous localization and mapping, and two of the most popular mapping ROS systems. In section 3.5 we describe the localization AMCL algorithm. We conclude with a note about the ROS reference system convention, in section 3.6

- In chapter 3 we start to write about the actual work, starting with 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 outputs. In particular we describe experiments about odometry estimation, in section 5.2, about mapping in section 5.3 and 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 in this field.

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 will introduce the Ra.Ro. platform, the robot we worked on.

Ra.Ro. stands for Ranger Robot, that is to say that its main purpose is to work as a security guard. However, the producer company, NuZoo offers the possibility of customizing the platform to meet different requests. In the following paragraphs we will focus on the version we worked on, introducing the hardware and software suite.

2.1 Hardware

Ra.Ro. is a skid steer drive robot. The base 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 stepper 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 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.

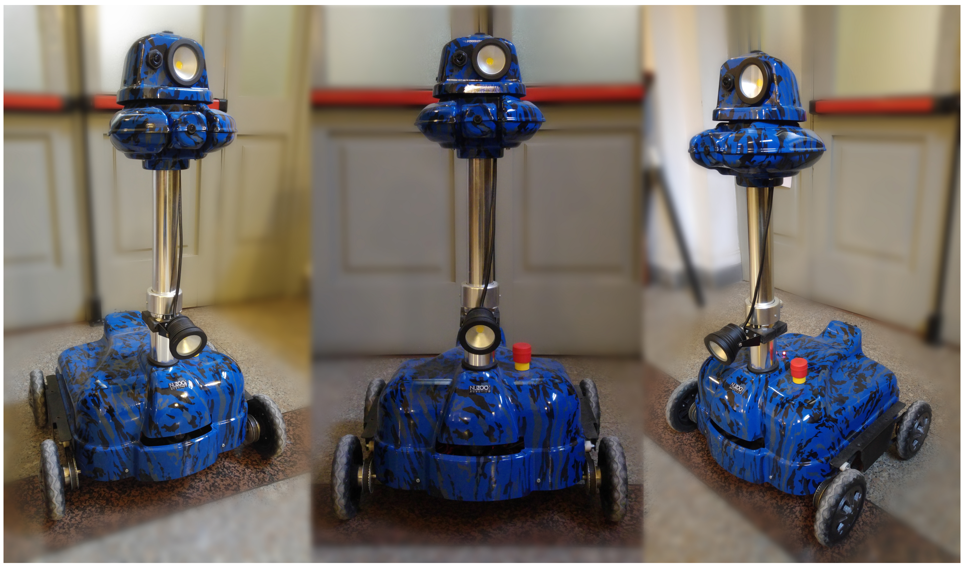


Figure 2.1: Three Ra.Ro. views

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. The cameras' vision into the darkness is guaranteed by two led flashlights.

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 needing, a wired connection is also available to recharge the robot.

2.2 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, there are nodes responsible for publishing sensors data, such as 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 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 and the recognized marker, and to

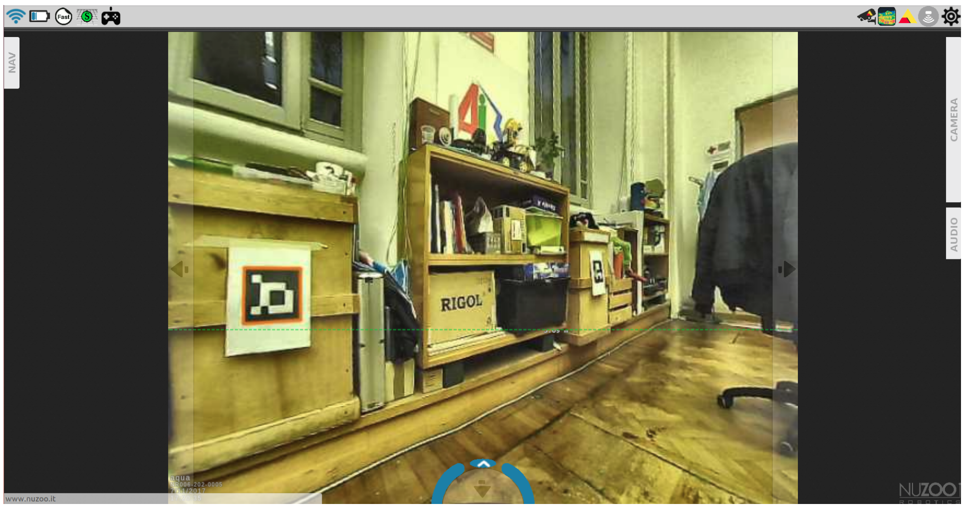


Figure 2.2: NuZoo web interface. The recognized marker is orange bordered.

manage the various minor functionalities such as speakers, lights and so on.

2.2.1 ROS topics

In this section we will describe the most relevant, for 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, which include gyroscope, accelerometer and magnetometer. The `r2p_msgs/ImuRaw` is a custom message type built by a 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 dps, with a sensitivity of 17.50 mdps per LSB 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 the gravitational acceleration which measures about 9.81 m/s^2

- `/r2p/odom`, `geometry_msgs/Vector3`:
  In this topic is published a very raw odometry, built in the board system. 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`:
  In this topic is published 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`:
  In this topic is published the odometry, built by integrating the gyroscope measurement in a ad hoc way. We improved this odometry for these project, as we will explain in **??**. 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`:
  In this topic are published messages from the Hokuyo laser scanner, after being filtered by some outliers. The `sensor_msgs/LaserScan` messages represent the collection of the distances at which an infrared ray beamed by the laser scanner has been 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

Figure 2.3: Ra.Ro. photograph from NuZoo website

close range. The teleoperation system operates in three ways: a manual one, an assisted one and a semi-autonomous.

Manual teleoperation

The manual teleoperation 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 is relies completely on human control.

Assisted teleoperation

The assisted teleoperation is managed via the web application interface which in this case can be runned 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 any obstacle in the trajectory. We can also consider as assisted teleoperation the one with the particular *follow me* marker. The operator must show the marker to the robot's navigation cam and the robot it will try to keep the marker into the camera frame, following the person that holds 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, if desired.

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.

2.3 Platform purposes

As introduced at the beginning of this chapter, the full Ra.Ro. platform is designed to be a security guard, able to patrol parkings, supermarkets and so on. It has been already 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 base.

Chapter 3

Knowledge requirements

"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 the state of the robot itself. 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 [9].

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 update frequently the estimate of the state and the past states as well while required.

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

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

3.1.1 Baysian state estimation

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

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

So the belief of a state in 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 retrieve the measurement of the sensor at the t time is not very practical. In fact, 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 recursive fashion, considering the first term as prior and the second one the posterior. The most general form of the recursive state estimation is the Bayes filter, which is here reported.**??**

For each state variable we can divide its estimation into two steps.In the first one, the *prediction*, we estimate the state variable at time t given the u_t and x_{t-1}. In the second one, the *innovation*, 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: $P(A) = \sum_n P(A|B_n)P(B_n)$
- 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 observation 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 mostly impossible to use since the analytical representation of the multivariate posterior is usually difficult to retrieve. 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 distributions and measurement noise follow a Gaussian distribution and that system and observation models are linear [6]. 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 in a sense the non-linearity of the system. The solution lies in Taylor series expansion applied to linearize the requested functions. These solutions are still widely employed today and are often the first choice in recursive state estimation. However, none of the KFs hold 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) which, using the *unscented transform* for the linearization, can obtain better results in terms of accuracy, keeping the characterization of the error as 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 ones does not rely on an analytic representation of the posterior probability distribution, nor on parameters or statistics that can represent them.

A well-known approach is the *particle filter*, proposed by Gordon et Al. in 1993 [4]. 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 still problems exist. 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

For SLAM problem, in 1997, Lu and Milios developed and proposed a 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 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, 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. The graph technique implies that not only the latest state can be estimated, but also the previous ones, making possible to continuously estimate the full robot trajectory.

An even wider generalization of the graph approach is the *factor-graph*, which is basically a hypergraph in which edges do not incide only between two nodes, but can possibly affect many of them. This comes up with a powerful tool in multi-sensor fusion problems, in particular it is appreciated the possibility to represent heterogeneous measurement, in sense of number of poses effected, maintaining a quite explicit design of the graph.

3.2 Odometry estimation

Odometry basically 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 piece of information given by the wheels' rotation is the base for the whole estimation, for every wheeled mobile robot.

3.2.1 Generic odometry

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

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

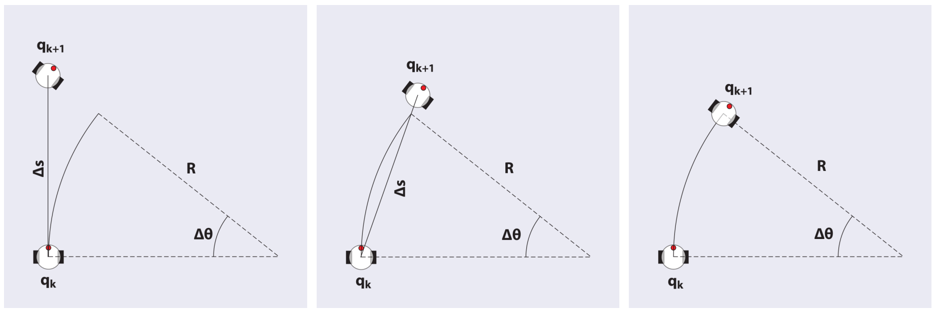
Each wheeled mobile robot (WMR) to be able to move must have a point around which all wheels follow a circular course. This point is known as the instantaneous center of curvature (ICC) or the instantaneous center of rotation (ICR). In practice it is quite simple to identify, because it must lie on a line coincident with the rotation axis of each wheel that is in contact with the ground. Thus, when a robot turns 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 in a t instant of time and $\omega(t)$ the angular velocity at the same time, can be generally retrieved the position and orientation of the robot at time t_1 as follows

$$\begin{aligned} x(t_1) &= \int_0^{t_1} v(t)cos(\theta(t))dt \\ y(t_1) &= \int_0^{t_1} v(t)sin(\theta(t))dt \\ \theta(t_1) &= \int_0^{t_1} \omega(t)dt \end{aligned} \tag{3.1}$$

In order to deal with concrete cases, namely discrete time, exist different integration methods. We will present three of the most common ones: Euler method, II order Runge-Kutta method and the precise reconstruction. For



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

Figure 3.1: Different integration methods results

these explanation we will use the relaxed notation $x_k = x(t_k), v_k = v(t_k)$ and so on, and we define $T_s = t_{k+1} - t_k$, namely the sampling period.

Euler method

$$\begin{cases} x_{k+1} = x_k + v_k T_s cos\theta_k \\ x_{k+1} = y_k + v_k T_s sin\theta_k \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases} \tag{3.2}$$

This is the most simple integration method, but also the most subject to error in x_{k+1} and y_{k+1}. θ_{k+1} is exact in fact 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

$$\begin{cases} x_{k+1} = x_k + v_k T_s cos(\theta_k + \frac{\omega_k T_s}{2}) \\ x_{k+1} = y_k + v_k T_s sin(\theta_k + \frac{\omega_k T_s}{2}) \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases} \tag{3.3}$$

Comparing with the Euler method, the II order Runge-Kutta one decreases the error in computation of x_{k+1} and y_{k+1} using the mean value of θ_k. Also in this case the smaller is sampling period T_s, the smaller is the error.

(a) TurtleBot

(b) Khepera

(c) Robuter

Figure 3.2: Commercial differential drive robots

Precise reconstruction

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

Here is involved the instantaneous radius of curvature $R = \frac{v_k}{y_k}$. Note that for $\omega_k = 0 \rightarrow R = \infty$ the equation degenerate matching the Euler and Runge-Kutta algorithms. The method is based on geometrical considerations.

3.2.2 Differential drive odometry

The differential drive mechanism consist in two active wheels, rotating on a common axis, drove by two different motors. In addition one or more passive castor wheel(s) can support the robot stability without interfere with the robot kinematic. Many commercial robots adopts this kind of mechanism due the simplicity of implementation, the relative low cost and the compactness of the system. We can found it in TurtleBot, Khepera or Robuter. The whole robot motion is based on the difference in rotation velocity of the

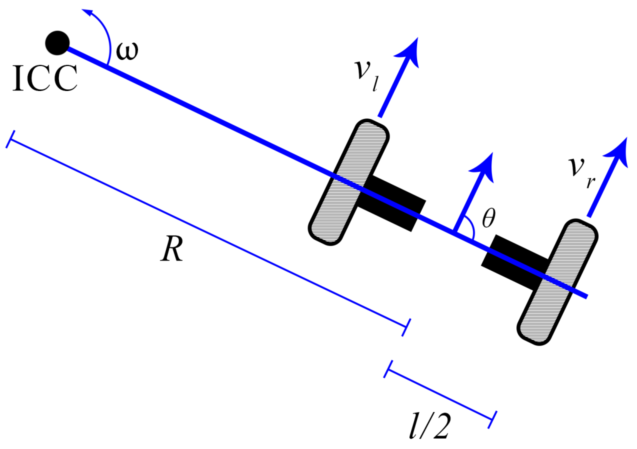


Figure 3.3: Differential drive kinematics

two active wheels. The ICC lies on of course on the axis the wheels rotate around and the curve that the robot will follow depend on the position of this point with the respect of the middle point between the two wheels.[IMG]

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 ω is given by the following

$$\omega(R + \frac{l}{2}) = v_r$$

$$\omega(R - \frac{l}{2}) = v_l$$

where l is the distance between the 2 wheels.

We are actually interested in retrieve R and ω" from the velocities, which are usually available data given by the encoders, and l which is a fixed parameter. So the formulas are:

$$R = \frac{l(v_r + v_l)}{2(v_r - v_l)} \qquad \omega = \frac{v_r - v_l}{l} \tag{3.5}$$

It is interesting to analyze a couple of special case. When $v_r = v_l$ then $R = \inf$, which means that the robot is moving in a straight line, is not curving. When $v_l = -vr$ then $R = 0$, which means that the robot is only rotating around the vertical axis passing through the midpoint between the wheels.

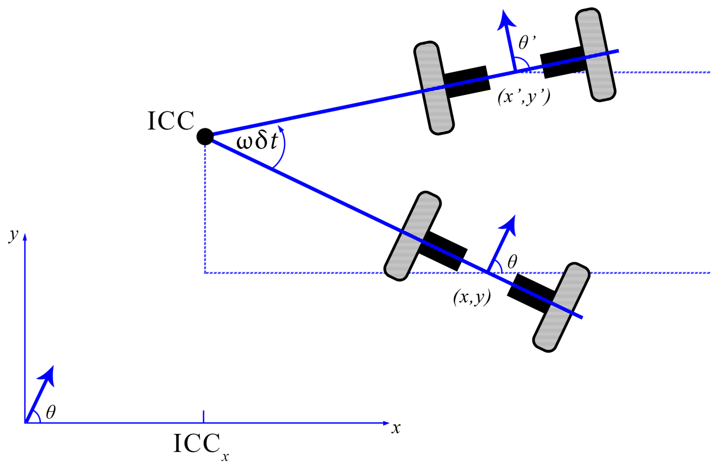


Figure 3.4: Differential drive robot motion from pose (x, y, θ) *to* (x', y', θ')

Since the differential drive structure is non-holonomic, is not possible to do, for example, a lateral movement or any other kind of displacement not represented by the previous equations.

The position (x', y', θ') in a particular instant of time is given by the computation on the previous (x, y, θ) and the time span Δt between the two positions

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

Where ICC_x and ICC_y are the coordinate computed as follows:

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

The specific case for the odometry calculation in differential drive, according with 3.1 is

$$\begin{aligned} x(t_1) &= \frac{1}{2}\int_0^t (v_r(t) + v_l(t))cos(\theta(t))dt \\ y(t_1) &= \frac{1}{2}\int_0^t (v_r(t) + v_l(t))sin(\theta(t))dt \\ \theta(t) &= \frac{1}{l}\int_0^t (v_r(t) - v_l(t))dt \end{aligned} \tag{3.6}$$

3.2.3 Skid-steering odometry

The skid-steering mechanic tries to maintains the simplicity and compactness of the differential drive, improving in the meanwhile the robustness of the model. It consists in four wheels that we can divide in 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, in addition to the higher robustness, with the respect of the differential drive, makes this kind of mechanic an optimal choice for different purposes. This mechanic, indeed, offer a good mobility on different terrains, not only indoor ones, but also outdoor, because one of the advantages is the possibility of mount 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, that was possible to use for the differential drive case, is no more an option because 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 indeed.

Wang et al. help to partially resolve the problem of the complex skid-steering kinematic proposing an approximation to the differential drive one. It is based on three assumption:

(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 running on a firm ground surface, and four wheels are always in contact with the ground surface

Then considering the figure and the deriving the Equations **??** we obtain

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = f \begin{bmatrix} w_l r \\ w_r r \end{bmatrix} \tag{3.7}$$

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

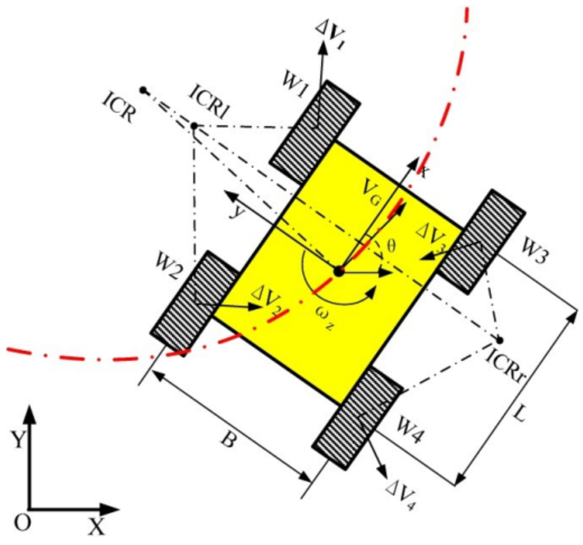


Figure 3.5: Skid-steering platform

During robot turn there are different $ICRs$: ICR_l, ICR_r and ICR_G, that are respectively of the left-side tread, right-side tread, and the robot center of mass. 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 the treads share the same angular velocity ω_z, so these equations follows

$$y_G = \frac{v_x}{w_z} \tag{3.8}$$

$$y_l = \frac{v_x - w_l r}{w_z} \tag{3.9}$$

$$y_r = \frac{v_x - w_r r}{w_z} \tag{3.10}$$

$$x_G = x_l = x_r = -\frac{v_y}{w_z} \tag{3.11}$$

from (**??**) to (**??**) the generic odometry kinematic (**??**) can be represented as:

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = J_w \begin{bmatrix} w_l r \\ w_r r \end{bmatrix} \tag{3.12}$$

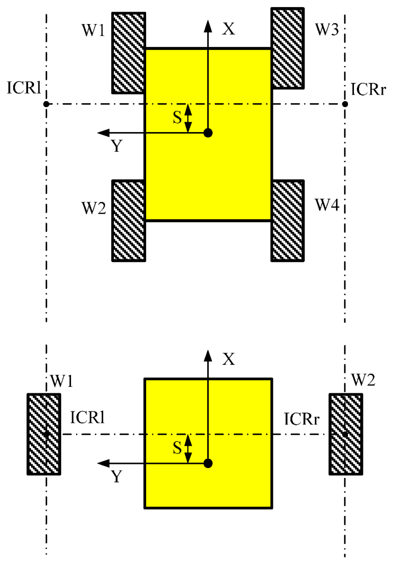


Figure 3.6: Skid-steering to differential drive equivalence

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

$$J_w = \frac{1}{y_l - y_r} \begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & 1 \end{bmatrix} \tag{3.13}$$

If the robot is symmetrical, then the $ICRs$ will lie symmetrically on the x-axis, and will be $x_G = 0$ and $y_0 = y_l = -yr$. The J_w matrix be rewritten as:

$$J_w = \frac{1}{2y_0} \begin{bmatrix} y_0 & y_0 \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \tag{3.14}$$

So the velocities defined in (**??**) can be defined, for the symmetrical model, as follows:

$$\begin{cases} v_x = \dfrac{v_l + v_r}{2} \\ v_y = 0 \\ w_z = \dfrac{-v_l + v_r}{2y_0} \end{cases} \tag{3.15}$$

and from (**??**) we can get the instantaneous radius of the path curvature:

$$R = \frac{v_G}{w_z} = \frac{v_l + v_r}{-v_l + v_r} y_0 \tag{3.16}$$

The ratio of sum and difference of left and right wheels' linear velocities can be defined as a variable λ

$$\lambda = \frac{v_l + v_r}{-v_l + v_r}$$

and (3.16) becames

$$R = \lambda y_0$$

A similar approach is used in Mandow's work, in which an *IRC* coefficient χ is defined as:

$$\chi = \frac{y_l - y_r}{B} = \frac{2y_0}{B}, \qquad \chi \geq 1 \tag{3.17}$$

χ represents the approximation from the differential drive kinematic. We can notice that when χ is equal to 1 there is no slippage and the skid-steering model coincide with the differential drive. It implies that we can use the differential drive model to approximate the skid-steering one, working on the χ variable. In particular the skid-steering coincide with a differential drive with a larger span between the left and right wheels as shown in Figure. This is a very useful method and will be very appreciated for our work.

3.3 Sensor fusion framework

Since, as mentioned before, the pure kinematics equations are actually just an approximation of the real world, they are not enough to retrieve the correct odometry of a robot with a satisfying precision. This is why the 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 stays 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. The main goals of the project include ensuring that the resulting software framework can be employed on very different robotic platforms and hardware sensor configurations and that it can be easily tuned 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 for 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 estimate of sensor parameters.

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* sensor, which are independent from the actual hardware that produce the measurement. For example an odometry measurement can be retrieved from a laser scanner elaboration or an 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 for its domain. This means that we must initialize the sensors passing the specific parameters for that sensor. For example a *DifferentialDriveOdometer* requires the wheels radius and the wheels distance, expressed in meters. Another required parameter needed to properly set up a sensor is its position and orientation with the respect of the tracked base frame, in order to properly use the information retrieved. For example the camera sensor, possibly used to retrieve markers positions, must be properly set in order to calculate the exact transformation between the seen marker and the base frame.

The ROAMFREE's modularity and its ROS implementation make it a very powerful tools for sensor fusion purposes and the other side aims. 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 already written about the important features of graph-based approaches in paragraph **??** in a general way. The most relevant advantage is the possibility of menage high-dimensional problems in 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

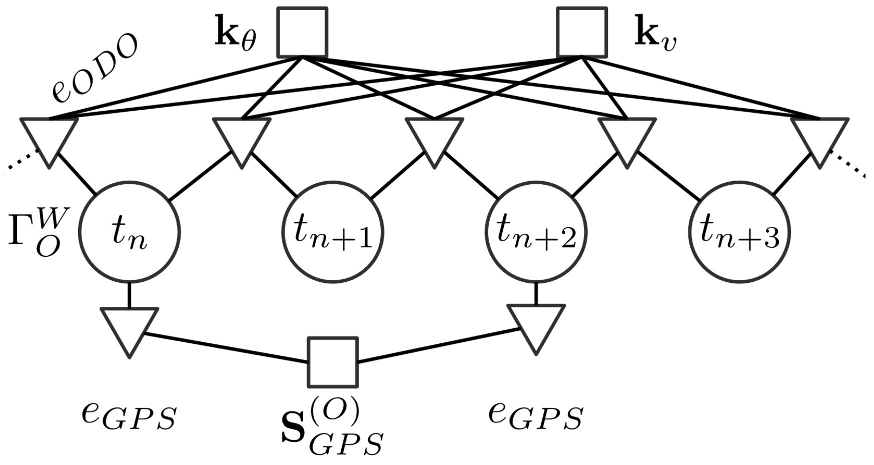


Figure 3.7: An instance of the pose tracking factor graph with four pose vertices $\Gamma_O^W(t)$ (circles), odometry edges e_{ODO} (triangles), two shared calibration parameters vertices k_θ and k_v (squares), two GPS edges e_{GPS} and the GPS displacement parameter $\mathbf{S}_{GPS}^{(O)}$

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 need one integration to retrieve a position; an acceleration needs three nodes because the twice integration required.

One, and only one, of the available sensors must be chosen as *architecture master sensor*, and a good practice is to choose an odometry sensor to play this role, because it has usually 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 begins a non-linear optimization process, using also the other measurement retrieved in the meanwhile. It can happens that sensor reading are late for low rate, connection problems or in general are not available. In these cases, if the available once are not sufficient to generate a pose, the measurement handling is delayed until enough data are available.

The new node generation is based on a fixed-lag window. It means that only nodes contained in these 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 holds in a new generated factor.

We can resume the factor-graph advantages here:

- Flexible with the respect of sensors' nature and number. The modularity of the system allows to manage all the inserted sensors in an independent and uniform way, by means 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 reading are available.

- Possible to deal with out-of-order measurements. If an old information is received, according to its time stamp, it will not be simply discarded, but will be created an appropriate factor, connecting the nodes interested by and updating and refining them.

- The quality of the estimation is higher and, in certain circumstances, faster than traditional filters, such as EKFs [8].

- The high degree of sparsity of the considered information fusion problem is explicitly represented and 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 these ones start from a common definition

$$e_i(t) = \hat{z}(t; \hat{x}_{S_i}(t), \xi) - z + \eta \tag{3.18}$$

where $hatx_{S_i}(t)$ is the extended state for the sensor frame S_i in which are represented the position and the orientation of the frame with the respect of the world frame, its position and orientation at time t; ξ 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 η is the zero-mean Gaussian noise representing the measurement uncertainty. It is evident that the more the prediction is accurate, even equal to z, 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 distinguish 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)

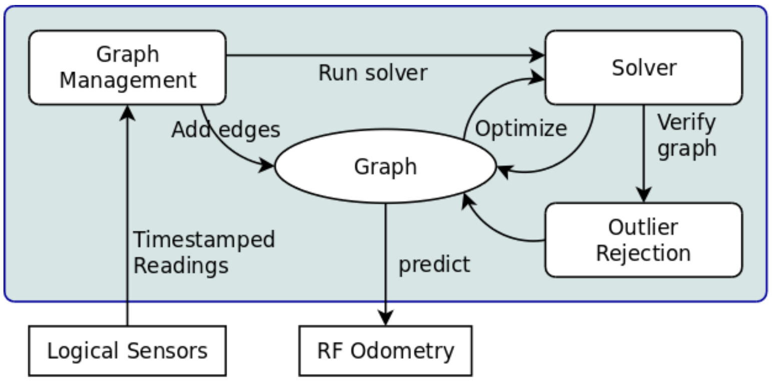


Figure 3.8: ROAMFREE estimation schema

(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 ξ, which includes gain, bias and other specific parameters according to the sensor class. These parameters sometimes are easy to retrieve from sensors' specifications or from observation, but often they need an accurate tuning to let everything works well.

An even more attention is required for covariance matrix setup. Indeed the factor graph wants this matrix as input in measurement update, in addition to the measurement itself and the sensor name. The covariance matrix can be even different according to the actual measurement, of course always being of the right dimension. Changing the value of the matrix means change the reliability of a sensor. In other words if two measurement indicate two conflicting outputs the system will *trust* more the one with a lower covariance.

A convenient feature is the outlier management made through the *robust kernel* technique. It consist 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 following computation. It is useful to manage 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 non-linear, wighted and least-squares optimization and here will discuss about how it is done. Consider the error function $e_i(x_i, \eta)$

associated to the i-th edge of the hyper-graph and defined as (**??**). We can approximate the error function as $\overline{e}_i(x_i) = e_i(x_i, \eta)|_{\eta=0}$ since e_i is a random vector. It can involve non-linear dependencies with respect to the noise, so its covariance Σ_η is computed by means of linearization, i.e.:

$$\Sigma_{e_i} = J_{i,\eta}\Sigma_\eta J_{i,\eta}^T|_{x_i=\breve{x}_i,\eta=0} \tag{3.19}$$

where $J_{i,\eta}$ is the Jacobian of e_i with respect to η 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 does not depend on x_i, for the i-th edge this function reads as:

$$\mathcal{L}_i(x_i) = \overline{e}_i(x_i)\Omega_{e_i}\overline{e}_i(x_i) \tag{3.20}$$

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

$$\mathcal{P} = \underset{x}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}_i(x_i) \tag{3.21}$$

It is possible to 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).

3.4 Simultaneous Localization and Mapping (SLAM)

With simultaneous localization and mapping (SLAM) is meant the building of a map while the robot locate itself into the map that is being created. The SLAM can be considered a preliminary phase in which the robot create the map and then it will use it for the autonomous navigation, or can be contextual to the autonomous navigation. It is important to point out that it does not matter if the robot, during the SLAM phase, is human controlled or not. The first localization guess is given by the odometry, but, as mentioned in section (**??**), 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 using world references observed through laser scanner(s), camera(s) (Visual SLAM) or similar sensors. It follow 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 is evident the possible problem that can shows up. For sure SLAM must be done in recursive way and this is one of the main reason why is a so complex task. From a probabilistic point of view, there are two main forms of SLAM. One is known as the online SLAM problem: it involves estimating the posterior over the momentary pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3.22}$$

Where x_t is the pose time 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. Many algorithms for the online SLAM problem are incremental: they discard past measurements and controls once they have been 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:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.23}$$

Instead of computes the state incrementally as in online case, here the sequence of states is computed one time.

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

(i) Landmarks detection: the robot must recognize some feature from the environment, called landmarks. For 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 landmark.

(ii) Data association: once the landmark is been detected, it must be matched with an 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 takes 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.

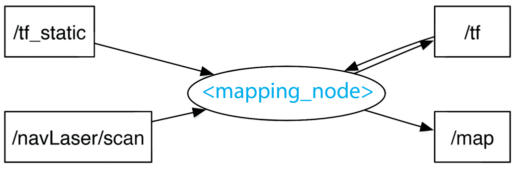


Figure 3.9: The image represent the topics subscribed and published by the mapping node, independently by the exact mapping system used

The major difficulties of SLAM are the following:

- High dimensionality: since the map dimension always 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 revisits a past place, the accumulated odometry error might be large. Then the data association and landmark detection must be effective to correct 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 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, because the probability to deal with a potential ambiguity is lower.

3.4.1 Gmapping

Gmapping is the most widely used laser-based SLAM package in robots worldwide. The algorithm has been proposed by Grisetti et al.in 2007 and it is a Rao-Blackwellized Particle Filter SLAM approach.

We mentioned the Particle Filter in section **??** and here we will describe RBPF, which is an optimized version for the SLAM problems.

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
      $\langle x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} \rangle = s_{t-1}^{(i)}$
      $//Scan - matching$
      $x_t^{'(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$
      $\hat{x}_t^{(i)} = \mathrm{argmax}_x \, p(x|m_{t-1}^{(i)}, z_t, x_t^{'(i)})$
      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
  $N_{eff} = \frac{1}{\Sigma_{i=1}^{N} (\tilde{w}^{(i)})^2}$
  if $N_{eff} < T$ then
      $\mathcal{S}_t = \text{resample}(\mathcal{S}_t)$
  end if
```

We can start from (3.23) and factorize it as:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{3.24}$$

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.

The RBPF occupancy grid SLAM works as follows:

If new control data u_t from the odometry and a new measurement z_t

form the laser scanner is available:

1. Determine 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. Perform 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. Draw pose $x_t^{(i)}$ of particle i from the approximated Gaussian ditribution of the improved proposal distribution.
5. Perform update of the importance weights.
6. Update map $m^{(i)}$ of particle i according to $x^{(i)}$ and z_i.

The more detailed RBPF algorithm pseudo-code can be read in (**??**). The author proposes a way to compute an accurate distribution by taking into account both she 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 decreased since the uncertainty is lower, due to the scan matching process, improving the performance.

3.4.2 Cartographer

Another possible approach for SLAM problem is using graph-based methods. These ones use optimization techniques to transform the SLAM problem into a quadratic programming problem. The historical development of this paradigm has been focused on pose-only approaches and using the landmark positions to obtain constraints for the robot path. The objective function to optimize is obtained assuming Gaussianity. Since this methods are based on a factor graph, they are able to remember better the old sub-maps and the old localization and thus results more accurate respect to other approaches.Their main disadvantage is the high computational time they take to solve the problem, so they are usually suitable to build maps off-line.

Google's Cartographer provides a real-time solution for 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 works only on submaps the error of the pose estimation in the world frame eventually increase. For this reason the system runs periodically a pose optimization algorithm. When a submap is considered 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 submap, 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 when a location is revisited. This leads to the soft real-time constraint that the loop closure scan matching has to happen quicker than new scans are added, otherwise it falls 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's pose relative to a map of its environment. It has been defined as one of the most fundamental problems in mobile robotics [3].

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; it can happen that an even well localized robot is moved, with no information about this transportation, to a different location. It can seems a similar problem to the second one, but here we can not trust a measurement as consistent with a previous one, because the loss of information during the unexpected movement. Moreover it is useful to correct very bad localization problems.

In other words the localization problem consist in identify 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 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 position,

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:         $x_t^{(m)}$ = SampleMotionModelOdometry($u_t, x_{t-1}^{(m)}$)
5:         $w_t^{(m)}$ = 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
```

which, for the two dimensional mapping, is typically represented as a three dimension vector $x_t = (x, y, \theta)$ in which x and y indicate the position of the robot in the map plane, and θ 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 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 particle is retrieved using the KLD-Sampling (Kulback-Leibler-Divergence) [5] [7]. The AMCL algorithm is here reported [**??**]. 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 status as a set of particles $\mathcal{X}_t$. This filter implementation is able to deal with the global localization problem, the localization problem and the kidnapping problem. The AMCL is flexible with the respect of the resampling technique, it 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 **??**). An AMCL ROS package is available [1] and a lot of robots uses 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 provides 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 [**?**].

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 point of reference. A right-handed chirality with `x` forward, `y` left and `z` up is preferred.

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, but drift makes it a poor frame for long-term 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) [?].

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 follows:

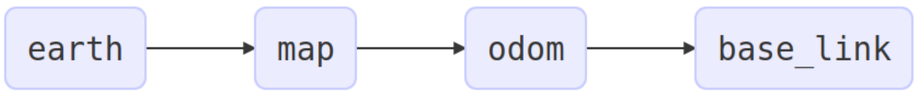


Figure 3.10: The tree frame representation.

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.

Frame Authorities

The transform from `odom` to `base_link` is computed and broadcast by one of the odometry sources.

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

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 kind of "autonomous" navigation system. It is based on line following and marker recognition, used as indication giver, such as "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 example, to indicate any point on a map and expect that the robot will reach that point. So, our aim was to reach a solution at least reliable as the previous one, but more powerful. Ra.Ro. is ROS based, so the most logic approach was to implement the ROS navigation stack, and to work around it. The best advantage from this approach is the modularity of the system, and here follows the explanation of the modules we used in our project:

Optimizations

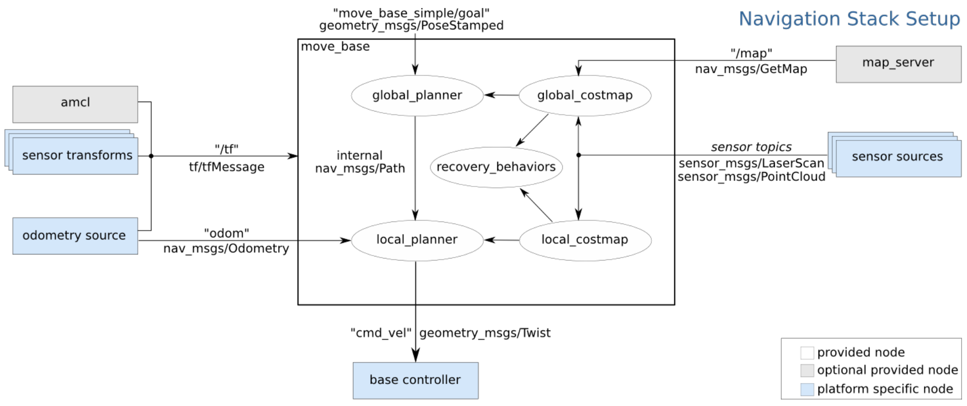


Figure 4.1: The ROS standard navigation stack schema

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 encoder, but it is usually very imprecise because the slippage of the wheels, different floors friction, small obstacles that let the wheel rotate without robot movement, imprecise sensor itself and so on. So, to have a better odometry source is a good choice to use a multi-sensor fusion system, and it is basically most of the work of this thesis.
We compared a custom source developed manually 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, the markers recognition.

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 one as sensor source. A future work on this project could add as sensor source a module that can extrapolate point cloud from cameras.

Sensor transforms

For each sensor 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 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 have to correct the position of the odom frame. During the initial phase, in which the robot doesn't have to navigate autonomously, the amcl module can be missing.

Map_server

As the amcl module, the map_server is optional because is used in the autonomous navigation phase. It consists in 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 that 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 into the whole map.

Local Costmap and Local Planner

The local costmap is similar to the global one, but instead of deal with the whole map, 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 outputted 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 modules

This the module is basically the most important for localization, mapping and autonomous navigation, since it is involved in all of these processes. In chapter ?? 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. suite projected for indoor environment, 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 an initial phase of the project it was developed an ad hoc odometry modifying the already existing Ra.Ro.'s code that was producing the odometry only from wheels encoders. This modification consists in implementing the Runge-Kutta integration method (??) and the precise reconstruction (??), instead of the Euler method (??), already implemented. Morover, instead of the θ_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, and then we have to frequently update the bias 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 it is true, it updates the gyroscope bias using a low-pass filter, in order to manage the sensor noise. 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;
```

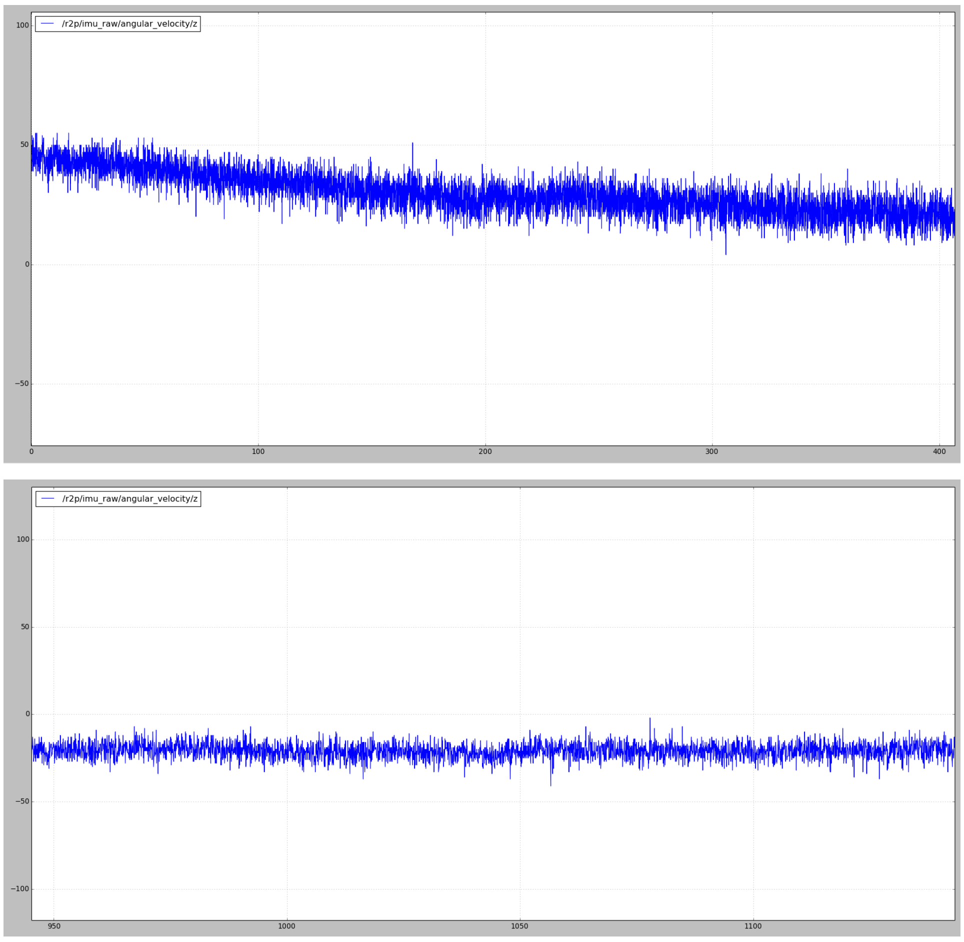


Figure 4.2: Gyoroscope measurement during two different time spans, with the robot stands

4.2.2 ROAMFREE module

The most reliable result we had is based on the ROAMFREE framework, introduced in **??**. The final set up is based on measurement given by the encoders, the gyroscope and the accelerometer and eventually 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 the markers improvement. The resulting `odom_r` frame is published in `/tf` topic. This node represent 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 influences the goodness of the estimation, but are also hardware depending, intended as computational power availability. Another feature of this node is the possibility of publish 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 republish messages read in r2p's topics and modified adding a header containing, above other header's info, the time stamp. It is useful to synchronize the left and right encoders' 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 + Encoders setup

One of the most reliable configuration we finally showed up is based on the data received from gyroscope, accelerometer in addition to the wheels' encoders ones. The 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 **??**. Thus, instead of pass the real distance between the wheels as parameter we passed a value

$$L' = L \cdot \lambda$$

where α has been experimentally calculated as

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

where ω_{real} is the real angular velocity, retrieved by the OptiTrack system, which provides the ground truth, and $\hat{\omega}_{diff}$ is the differential drive estimated angular velocity, retrieved by differential drive equations.

We used as *AngularVelocity* sensor the raw information given by the IMU sensor, as copy of the MEMS memory, properly scaled. We should have used as scale factor the one provided by the sensor specifications manual, but, since errors can happens in sensor production phase, we estimated this factor experimentally, as well. We obtained a slightly different value, which has been used to describe more accurately the real angular velocity.

Because of the difficulty in real acceleration retrieving, we had to use the specifications information for the *LinearAcceleration* sensor, i.e. the accelerometer.

Adding markers recognition

In addition to the other sensor we introduced the possibility to hopefully have a more accurate odometry estimation by means the marker visualization. As mentioned before, 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 of 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 the marker sensors as *FixedFeaturePosition* in order to not 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 absolute marker position as constant parameter. The sensor frame position is supposed to be fixed with the 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.

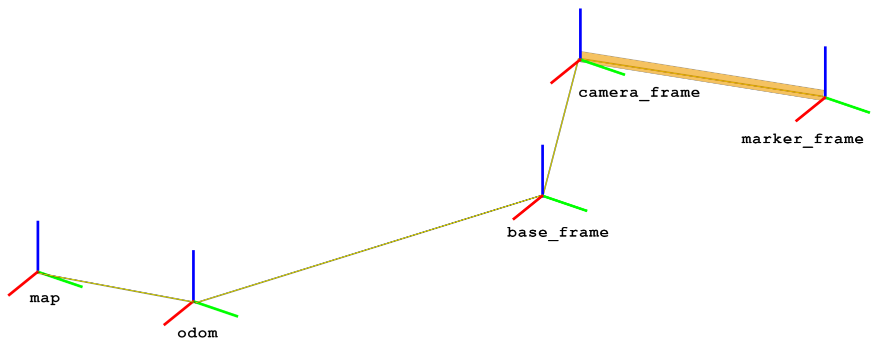
Because the extent of the environment we worked on, the markers real positions were not easily to retrieve, moreover we think that the solution we used introduces a desirable flexibility. We decided to initialize a new marker sensor as soon as the first marker observation, for a specific marker id, is done. It implies that the first marker observation should be done with a odometry error as small as possible, in order to not 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 in long distance covered. So the most desirable case is to see the marker as soon as possible, with the odometry estimation error hopefully small, set the marker sensors, and then using the set marker to keep the odometry estimation as correct as possible.

The marker visualization, as here presented, cannot be used if a node moves the odom frame in order to improve the localization, as explained in **??**, 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.

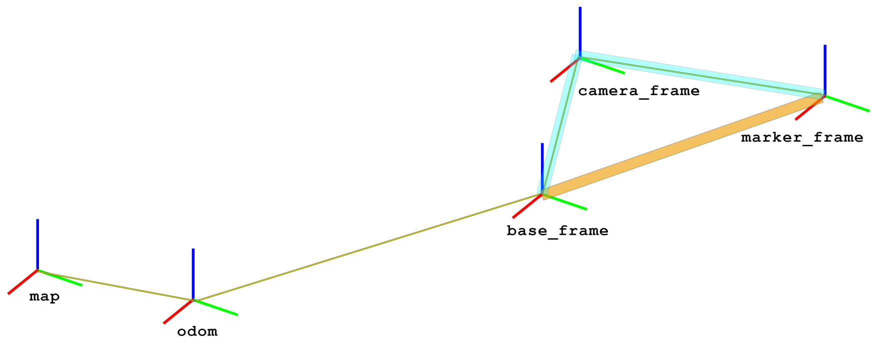
We solve this problem in a way that can seems counterintuitive, but is actually exact and experimentally confirmed. The idea is to pass as the marker observation measurement the composition of the map-odom transform and base-marker transform. In this way we inform ROAMFREE about the transform *behind* 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

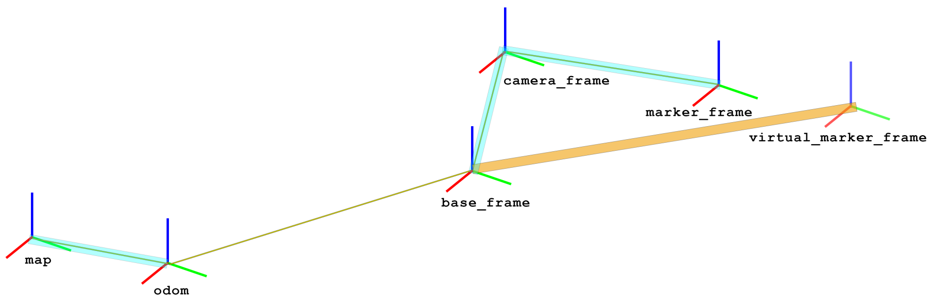
At high level the mapping node does not depend on the algorithm used to do the SLAM process. Both gmapping and cartographer, the two systems



(a) The transform required by ROAMFREE



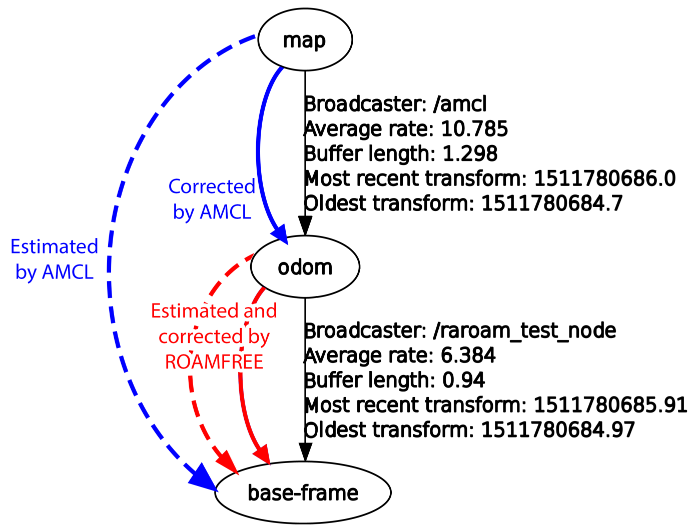
(b) The transform passed to ROAMFREE, setting the base frame as sensor frame



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

Figure 4.3: Different marker frame measurement passed to ROAMFREE. The ones blue highlated are the tranformation required and composed, the orange ones are the resulting transforamations

used here, takes 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



map
Broadcaster: /amcl
Average rate: 10.785
Buffer length: 1.298
Most recent transform: 1511780686.0
Oldest transform: 1511780684.7
Corrected by AMCL
Estimated by AMCL
odom
Estimated and corrected by ROAMFREE
Broadcaster: /raroam_test_node
Average rate: 6.384
Buffer length: 0.94
Most recent transform: 1511780685.91
Oldest transform: 1511780684.97
base-frame

Figure 4.4: The tree frame representation, with node explanations

odometry estimation. This movement is done by applying a estimated error rigid transformation to the odom frame. The **??** can explain better the concept of the relationships between the frames and the nodes that correct them. The AMCL node works between the same nodes as the mapping nodes, as we will explanin in 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. We was 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 have not included experiments about it in the **??**. 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, seldom losing the localization.

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 follows some details about the three nodes involved in 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, path planning between that goal and the estimate position of the robot and sending the actual velocity command for the robot movement. It has to deal, in order to create the path, with the set global and local costmaps.

Chapter 5

Experiment

“ [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 will illustrate the most significant cases in which we tested our system. We are going to introduce the general environment setup for the experiments and then we will 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 must be added system topics:

– `/rosout`

– `/rosout_agg`

– `/clock`

We applied the `rosbag` system for at least two reasons. The first one is the classical practical reason. Since a long parameters' tuning was needed, 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 other 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 example, for mapping phase, the ROAMFREE system, with specific configurations, can be computationally heavy, so a slower replay is appreciate. It can be acceptable even for a production phase because the robot can easily do the exploration for mapping purposes 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.

The Polimi's AirLab is provided of the OptiTrack - Motion Capture System, which gives the possibility of retrieve the ground truth in the covered area. It is an area around 5m $\times$ 5m wide; not so much, but can be significant.

5.2 Odometry experiments

In this section we will focus on the experiments involving only the odometry estimation. In particular we will 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:

- Start at the black square into the AirLab.

- Have a round into the room and then go out through the door.
- Go right until the end of the corridor is reached.
- Turn around an go back until the vertical corridor is reached.
- Follow the corridor and back again.
- Continue to the left corridor moving through the big room big room and go inside the small room connected to the big one.
- Continue to the left corridor until the end is reached and then go back following an almost straight line.
- Go again through the vertical corridor for a little and then come back to the AirLab.
- Finish the ride in the same point we started.

We ran the same `rosbag`, each time with one of these configurations. 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 one. We wants 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 Odometry with custom odometry

For this experiment we used only the custom odometry. Trying to fit the path generated into the floor plan we can notice that it is a quite good results. We can notice that the path generated is almost straight along the corridors and for the return part we can see that, when we drove the robot inside the vertical corridor again, the path are very close to match. The final point diverge form the initial one for about 2 meters, out of the more than 100 meters traveled.

5.2.2 Odometry with ROAMFREE odometry

For this experiment we set ROAMFREE as presented in **??**. 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 became too much bended, probably due to a bad gyroscope bias estimation, which persists for all the following part of the ride, finishing far from the initial point. It is necessary to point out that replaying the same bag we

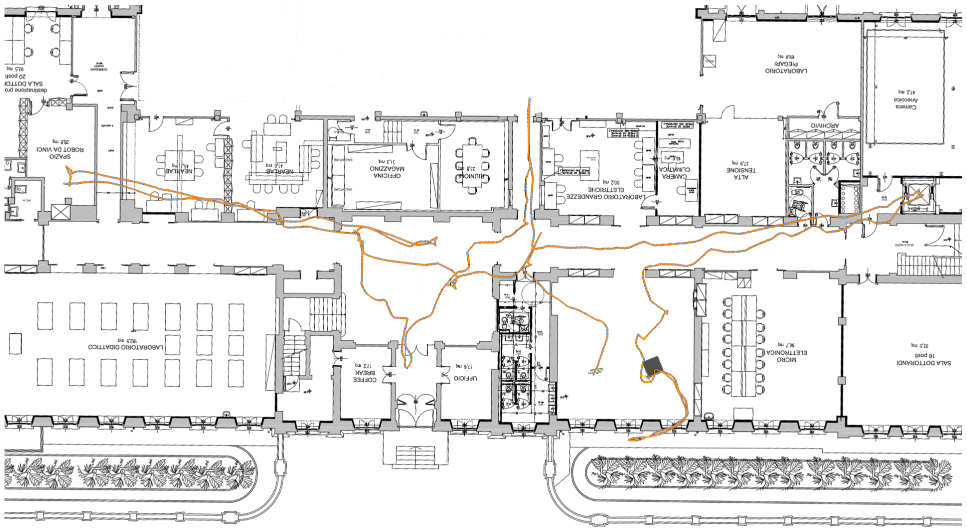


Figure 5.1: The resulting path generated with the custom odometry

can occur in different results. This can happen because the heavy mathematical computations done by the framework which sometimes need some approximation which can became significant after various multiplications.

5.2.3 Odometry with ROAMFREE odometry and markers

This configuration is evidently the best one. We set up the ROAMFREE environment as the previous one, except the addition of marker sensor as presented in **??**. We 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. It implies that a good odometry, without markers, is necessary. Moreover, the marker measurements improve the estimation of the gyroscope bias made by ROAMFREE, improving the odometry estimation itself. Markers add to the system the missing fixed point reference that we needed because the GPS unavailability, giving a very good odometry estimation, finally. It is important to point out that we used markers as fixed point, 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 picture **??** we can also see the generated markers constellation.

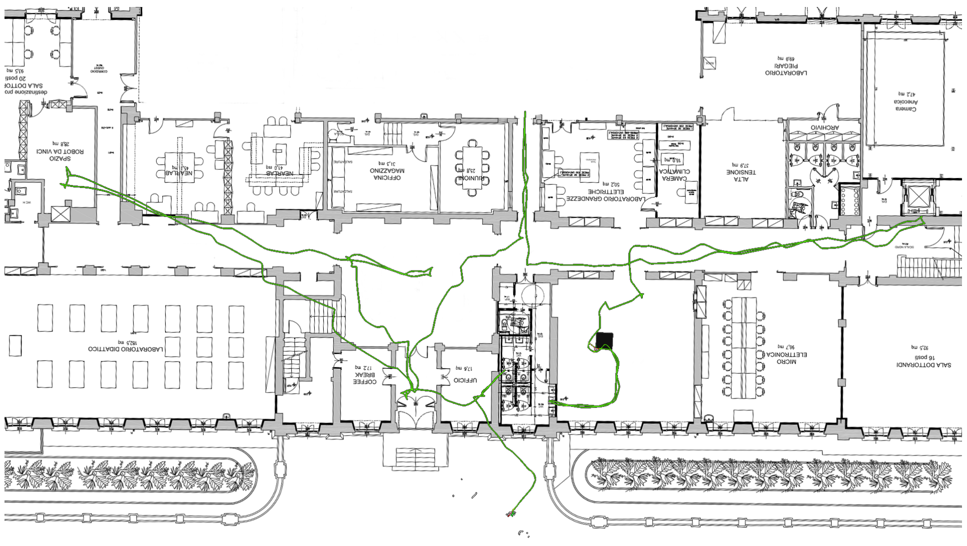


Figure 5.2: The resulting path generated with the ROAMFREE odometry, without markers

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. Instead, we could set up Gmapping for this purpose and the most significant parameters are reported in **??**.

5.3.1 Mapping with custom odometry

Using the custom odometry we retrieve a good enough map, slightly bended along the long corridors, which is a typical error in mapping. To notice, instead, that the length of corridors is very similar to the real ones, even though the common problem in mapping in long corridors is their shortening.

For the set up we completely deactivated the ROAMFREE module and we launch 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

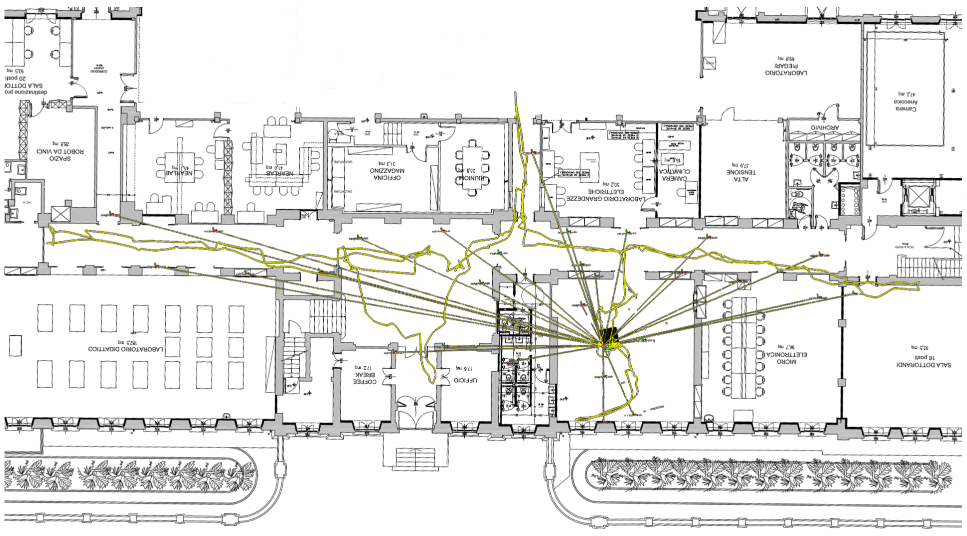


Figure 5.3: The resulting path tree generated with ROAMFREE odometry and markers

we leave both the odometry running because the poor modularity into the built in robot system, but we set the Gmapping system to run with the `base-frame` published by ROAMFREE.

5.3.3 Mapping with ROAMFREE odometry and markers

The map generated with the ROAMFREE odometry with markers included 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, causing eventually a bad 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.

5.4 Navigation experiments

We tested the navigation system selecting the map generated with the ROAMFREE including markers odometry and switching the estimation odometry system. Due the good map generated, in every case the possibly error caused by the odometry estimation can be well corrected by AMCL, which improves the localization into the map using the laser scans. Thus 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.

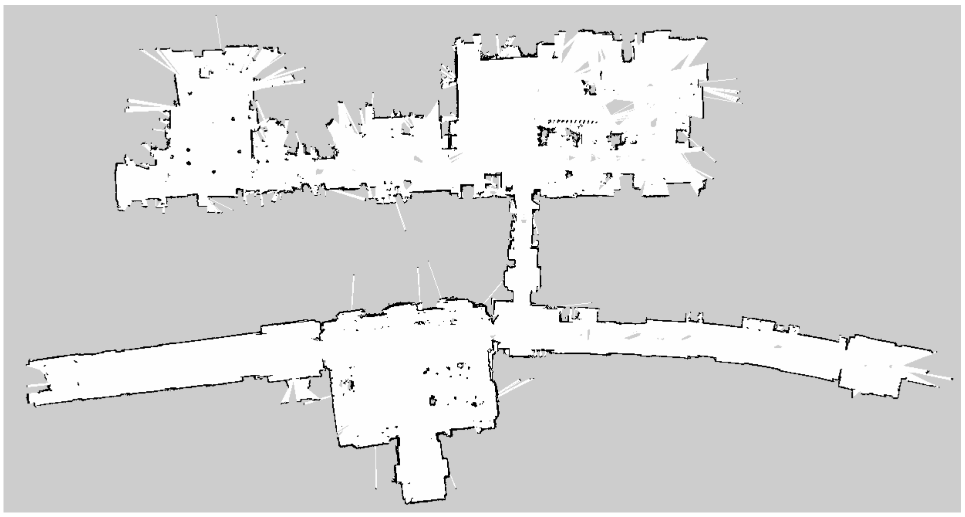


Figure 5.4: The resulting map generated with the custom odometry

The occurred problems during the navigation are caused by obstacles invisible to the laser scanner, causing the robot crushing into them and getting it stuck for a while. In some situation the recovery system, which consist basically in turning around the robot vertical axis finding laser features to match, was able to relocate the robot, no matter which odometry estimation system was involved.

ROAMFREE configuration		
Logical Sensor (sensor)	Measurement covariance	Note
`DifferentialDriveOdometer` (Encoders)	$\begin{bmatrix} 0.01 & \cdots & 0.0 \\ 0.0 & \ddots & 0.0 \\ \vdots & \cdots & \vdots \end{bmatrix}$	The matrix is 6 × 6 wide, with 0.01 diagonal values. This implies a quite high importance for the odometry retrieved from the encoders.
`AngularVelocity` (Gyroscope)	$\begin{bmatrix} 0.0001 & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0 \\ 0.0 & 0.0 & 0.02 \end{bmatrix}$	We decided to fix the `x` and `y` measurement to 0, being a differential drive, giving very high importance to these two component, and set the rotation around the `z` axis covariance to 0.02.
`LinearAcceleration` (Accelerometer)	$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0001 \end{bmatrix}$	In this case we decided to set the `z` component fixed to -9.81, i.e. the g acceleration, and set low accuracy for the other components, because the accelerometer is not very accurate as sensor, and difficult to manage because needs to be integrated twice

Markers in ROAMFREE configuration		
Logical Sensor (sensor)	Measurement covariance	Note
`FixedFeaturePosition` (Markers)	$\begin{bmatrix} 0.0001 & 0.0 & 0.0 \\ 0.0 & 0.0001 & 0.0 \\ 0.0 & 0.0 & 0.0001 \end{bmatrix}$	We imposed very high reliability on markers because we noticed a very accurate recognition.

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)

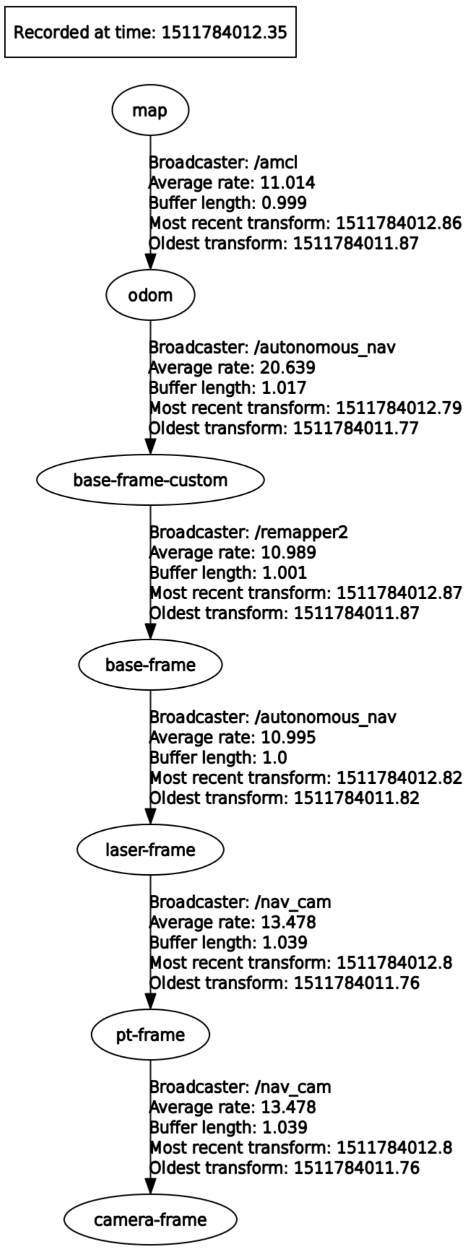

Recorded at time: 1511784012.35
map
Broadcaster: /amcl
Average rate: 11.014
Buffer length: 0.999
Most recent transform: 1511784012.86
Oldest transform: 1511784011.87
odom
Broadcaster: /autonomous_nav
Average rate: 20.639
Buffer length: 1.017
Most recent transform: 1511784012.79
Oldest transform: 1511784011.77
base-frame-custom
Broadcaster: /remapper2
Average rate: 10.989
Buffer length: 1.001
Most recent transform: 1511784012.87
Oldest transform: 1511784011.87
base-frame
Broadcaster: /autonomous_nav
Average rate: 10.995
Buffer length: 1.0
Most recent transform: 1511784012.82
Oldest transform: 1511784011.82
laser-frame
Broadcaster: /nav_cam
Average rate: 13.478
Buffer length: 1.039
Most recent transform: 1511784012.8
Oldest transform: 1511784011.76
pt-frame
Broadcaster: /nav_cam
Average rate: 13.478
Buffer length: 1.039
Most recent transform: 1511784012.8
Oldest transform: 1511784011.76
camera-frame

Figure 5.5: The resulting TF tree generated with the custom odometry

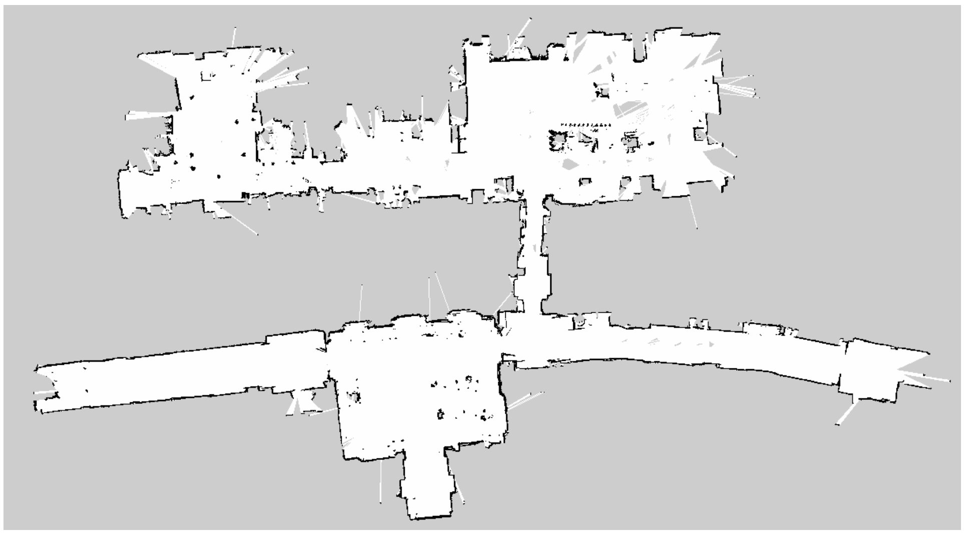


Figure 5.6: The resulting map generated with the ROAMFREE odometry without markers

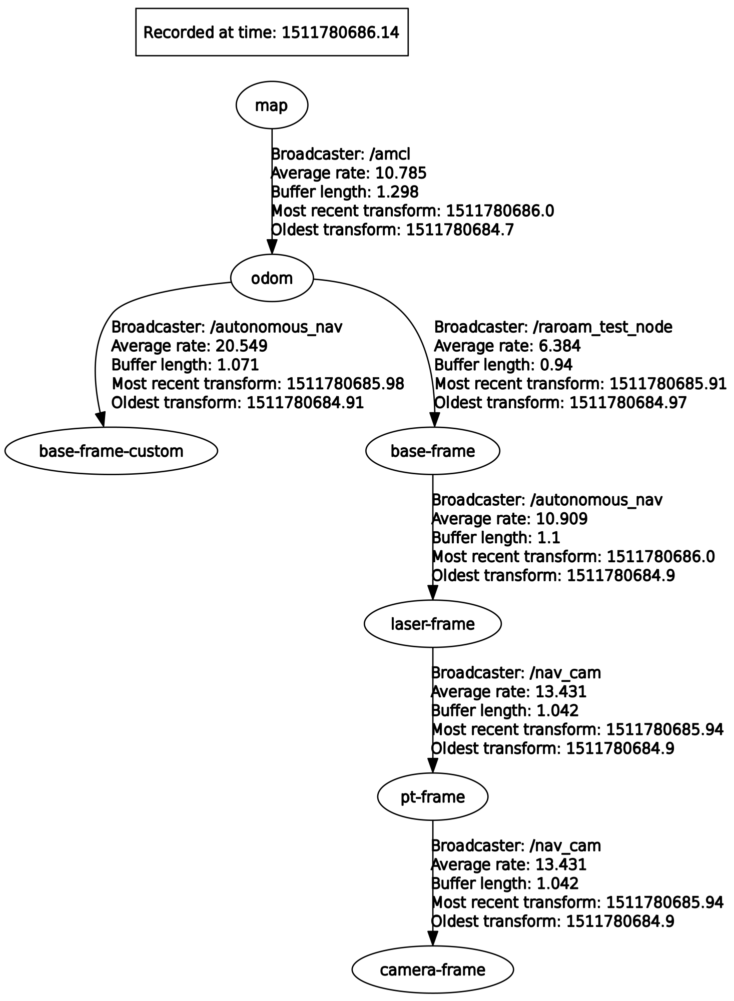

Recorded at time: 1511780686.14
map
Broadcaster: /amcl
Average rate: 10.785
Buffer length: 1.298
Most recent transform: 1511780686.0
Oldest transform: 1511780684.7
odom
Broadcaster: /autonomous_nav
Average rate: 20.549
Buffer length: 1.071
Most recent transform: 1511780685.98
Oldest transform: 1511780684.91
Broadcaster: /raroam_test_node
Average rate: 6.384
Buffer length: 0.94
Most recent transform: 1511780685.91
Oldest transform: 1511780684.97
base-frame-custom
base-frame
Broadcaster: /autonomous_nav
Average rate: 10.909
Buffer length: 1.1
Most recent transform: 1511780686.0
Oldest transform: 1511780684.9
laser-frame
Broadcaster: /nav_cam
Average rate: 13.431
Buffer length: 1.042
Most recent transform: 1511780685.94
Oldest transform: 1511780684.9
pt-frame
Broadcaster: /nav_cam
Average rate: 13.431
Buffer length: 1.042
Most recent transform: 1511780685.94
Oldest transform: 1511780684.9
camera-frame

Figure 5.7: The resulting TF tree generated with the ROAMFREE odometry without markers

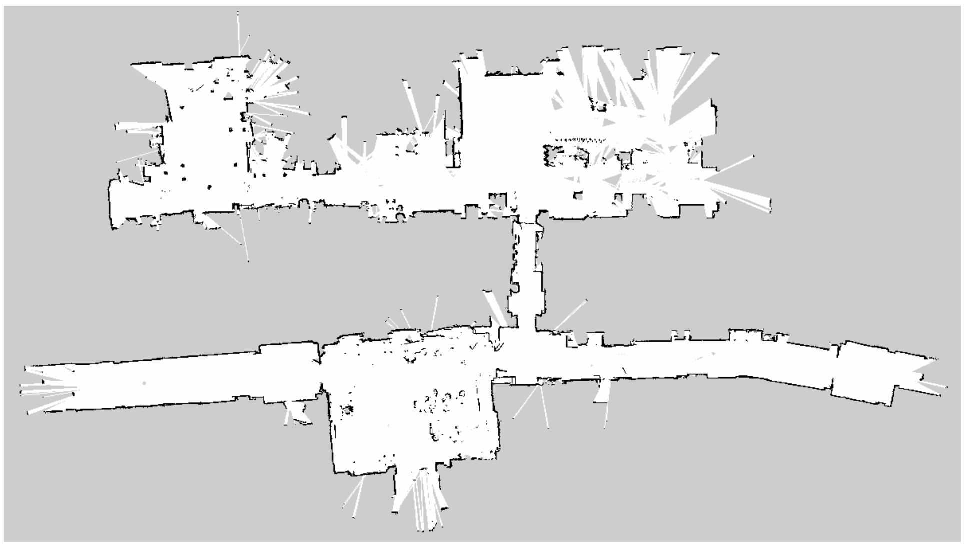
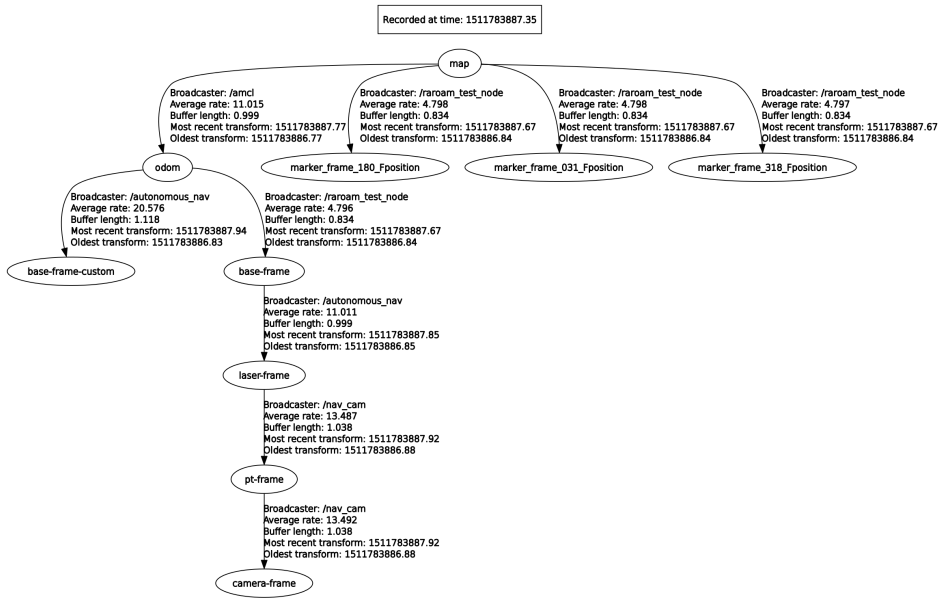


Figure 5.8: The resulting map generated with the ROAMFREE odometry with markers



Recorded at time: 1511783887.35
map
Broadcaster: /amcl
Average rate: 11.015
Buffer length: 0.999
Most recent transform: 1511783887.77
Oldest transform: 1511783886.77
Broadcaster: /raroam_test_node
Average rate: 4.798
Buffer length: 0.834
Most recent transform: 1511783887.67
Oldest transform: 1511783886.84
Broadcaster: /raroam_test_node
Average rate: 4.798
Buffer length: 0.834
Most recent transform: 1511783887.67
Oldest transform: 1511783886.84
Broadcaster: /raroam_test_node
Average rate: 4.797
Buffer length: 0.834
Most recent transform: 1511783887.67
Oldest transform: 1511783886.84
odom
marker_frame_180_Fposition
marker_frame_031_Fposition
marker_frame_318_Fposition
Broadcaster: /autonomous_nav
Average rate: 20.576
Buffer length: 1.118
Most recent transform: 1511783887.94
Oldest transform: 1511783886.83
Broadcaster: /raroam_test_node
Average rate: 4.796
Buffer length: 0.834
Most recent transform: 1511783887.67
Oldest transform: 1511783886.84
base-frame-custom
base-frame
Broadcaster: /autonomous_nav
Average rate: 11.011
Buffer length: 0.999
Most recent transform: 1511783887.85
Oldest transform: 1511783886.85
laser-frame
Broadcaster: /nav_cam
Average rate: 13.487
Buffer length: 1.038
Most recent transform: 1511783887.92
Oldest transform: 1511783886.88
pt-frame
Broadcaster: /nav_cam
Average rate: 13.492
Buffer length: 1.038
Most recent transform: 1511783887.92
Oldest transform: 1511783886.88
camera-frame

Figure 5.9: The resulting TF tree generated with the ROAMFREE odometry with markers

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, such as those we will write below, 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 been 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.

The physical hardware limitation we encountered, in fact common to many robot are the following:

- Bad laser scanner placement:
  The laser scanner is placed inside the robot body and the cover itself limits the laser scanner range, which makes the localization and the mapping phase harder. Moreover the LIDAR is placed about 10 cm from the floor, which is good because the robot can overcome maximum 10 cm high obstacles, nominally, but it introduce problem with tables and other similar object caused by the actual height of the robot, which is 75 cm.

- Bad mounted wheels:
  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 human almost subconsciously, but if the robot is driven by the artificial intelligence it is more complex and the result is a *swinging* trajectory.

Another main problem we had to deal with was the poor ROAMFREE documentation. We could count on the help of the author and of other people who previously worked with this library, but obviously is not a very feasible way, even less practical in a non academic environment.

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.

Bibliography

[1] Amcl documentation. Accessed: Jan 2017.

[2] Novalab website. Accessed: Jan 2017.

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

[4] 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.

[5] 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.

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

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

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