Sensor Fusion of Raw GPS Measurements for Autonomous Vehicle Localization

Author: **Pier Tirindelli**

Advisor: **René Alquézar Mancho**
from Department of Computer Science (UPC)
and Institut de Robòtica i Informàtica Industrial (CSIC-UPC)

Co-Advisor: **Andreu Corominas Murtra**
from Institut de Robòtica i Informàtica Industrial (CSIC-UPC)

Master in Artificial Intelligence

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
FACULTAT DE MATEMÀTIQUES (UB)
ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA (URV)

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)
UNIVERSITAT DE BARCELONA (UB)
UNIVERSITAT ROVIRA I VIRGILI (URV)

April 21, 2016

Summary

This dissertation outlines the benefits that sensor fusion brings in the mobile robotics problem of incrementally constructing a consistent map of an unknown environment with a moving robot, while simultaneously determining its location within this map.

The work of this Master's Thesis focuses on outdoor localization of autonomous vehicles, and aims to use raw GPS pseudorange data to establish geometric constraints to be added in the localization problem. This localization problem will solve for the optimal vehicle localization state, satisfying these pseudorange constraints, but also other geometric constraints imposed by other sensor measurements.

This project has been developed in the context of the European project Cargo-ANTS, devoted to investigate and develop novel techniques for autonomous navigation of wheeled vehicles in port terminal areas.

We conducted an introductory study on the GPS technology, focusing on the mathematical concepts used in the process of determining a position, and on the signals sent by GPS satellites, in order to be able to extract from raw data the geometrical constraints needed for the localization problem.

Each geometrical constraint depends principally from the range between the satellite and the receiver, and the position of the satellite in the moment when it sends this data. The GPS concept is based on time, and the range between the satellite and the receiver is calculated from the propagation time of the signal. In this calculation, we must take into account that there is an unknown bias between the receiver's internal clock and the one in the satellite. Pseudorange measurements need also to be corrected from errors introduced by ionospheric perturbations and other relativistic issues.

To estimate the position of the satellites, in the precise moment they sent the signal used by the receivers to compute the pseudorange measurements, we need to know all their orbital parameters. These parameters are contained in data blocks called ephemerides, and they are sent in different messages respect to the pseudoranges, and at a lower rate.

With the obtained data we are able to build a non-linear least squares

optimization problem that minimizes the sum of the squared differences between the received and expected range for each satellite observed in a determined time. The values found from this problem corresponds to the receiver three-dimensional position and the clock bias.

We applied the expertise acquired from this preliminary study to develop a software framework that permits to interface with a real GPS receiver, process the raw data captured with all the corrections needed, and build the optimization problem. To solve this optimization problem we use an external tool called Ceres-Solver.

We tested the developed software components acquiring real data in an urban environment and comparing the receiver three-dimensional position estimate with the GPS fix produced by the receiver. The obtained results are correct, in fact the estimated trajectory has the same shape of the one produced by the GPS fixes. The estimated trajectory is, as we expected, more noisy than the receiver's fix. This is due to the techniques implemented in the receiver's software to filter the results using the velocity of the receiver to predict the next position as long as other GNSS augmentations.

The second main part of this Master's Thesis project deals with sensor fusion in the SLAM problem. The Mobile Robotics Group of the institute where this Thesis has been carried out is developing Wolf, a C++ software framework to manage SLAM and sensor fusion. The philosophy behind Wolf is to automate the basic functionality needed in a SLAM problem, which is setting the geometrical constraints imposed by sensor measurements. This allows the developers to focus on the engineering of localization solutions and not on the low level coding.

The main structure in Wolf is a tree of base classes that describe the common elements of a SLAM problem: a robot, with its sensors, its trajectory formed by key-frames (state of the robot at a given time), and the map with its landmarks. These base classes can be derived to use Wolf in the particular SLAM problem needed.

Wolf provides also a set of sensor classes that are available or are being developed right now. We integrated in Wolf our work with GPS, extending the corresponding base class in order to describe the GPS sensor, with its intrinsic and extrinsic parameters. Each sensor needs a processor, that is a set of procedures used to operate on the raw data captured and extract geometrical constraints for the optimization problem. In the GPS case, each capture of data represents a different GPS observation, with all the satellite positions and pseudoranges observed in a precise moment. From each pseudorange measurement a constraint is created.

To define the GPS constraints we had to specify a set of coordinate frames that define the problem, as long as the GPS works in the ECEF coordinate

system and the robot refers to a map, and not to an absolute position. Four of these frames are necessary to correctly model our problem: the *ECEF* frame that refers to the center of the Earth; the *Map* frame, the one in which the robot is located and tries to incrementally construct and simultaneously determine its location within it; the *Base* frame, that represents the robot; and *GPS* frame, that is the antenna position with respect to the robot. In Wolf we have implemented all the math required to manage transformations between these frames and create GPS pseudorange constraints to the optimization problem.

The second part of this work has been implemented by developing a software package that fuses raw GPS measurements with the odometry using Wolf. In this step we had to add another coordinate frame to our structure, *Odom*, which represents the origin of the odometry trajectory. We tested this implementation doing an experiment using Teo, a four-wheeled Unmanned Ground Vehicle of the laboratory. Thanks to the sensor fusion, we are able to give an absolute positioning to the robot's trajectory, which is smoother than the trajectory estimated using only raw GPS. The GPS constraints remove the drift of the odometry, fixing one of the biggest flaw of odometry sensors. On the other hand, odometry adds robustness to the localization system in case of interruption of GPS data, due to sensor breakage, temporary obstructed line of sight with satellites or also poor reception, when only few satellites are available, and they are not enough to calculate a GPS fix.

In the end we draw future developments we could do to improve our solution, based on results of our experiments. In order to improve the quality of our implementation, the most important aspect we have to tackle is to enhance the outlier detection, to discard or improve faulty GPS measurements, especially taking into account multi-path interference in GPS signals.

Acknowledgements

I would like to thank the *Institut de Robòtica i Informàtica industrial* that gave me the opportunity to conduct this Thesis, as well as access to the laboratory and research facilities.

In particular, I would like to express my deepest gratitude to my co-advisor, Andreu Corominas Murtra. He has been always present and helpful, guiding me during the entire course of this project. I am also thankful to him for carefully reading and commenting on numerous revisions of this report.

My sincere thanks goes also to Joan Vallvé Navarro. His office door was always open whenever I ran into problem related to Wolf, and he helped me understand a lot of technical details about it.

I would also like to thank Joan Solà Ortega for the wise advices and clarifications about the SLAM problem and the Wolf architecture.

I am grateful to Sergi Hernández Juan, without his technical support it would have been impossible to do the experiments using the GPS receiver and the robot.

Contents

1. Introduction

1.1 Motivations and Objectives

Navigation is defined as the science of getting a craft or person from one place to another. In a broader sense, it can refer to any skill or study that involves the determination of position and direction. In every movements we make everyday we use some form of navigation and localization. All navigational techniques involve locating the navigator's position with respect to known locations or patterns. Modern techniques of navigation involve electronic sensors to perceive the surrounding environment and act accordingly to the information received.

One of the most widely distributed technology to achieve localization is the GPS. It is freely accessible all-over the world with cheap devices. However GPS presents some disadvantages that makes this technology not sufficient for all the applications and in every environmental situation.

To overcome this problem, different kinds of sensors can be used concurrently, originating localization through sensor fusion.

Mobile robotics is an area of robotics that heavily uses localization and navigation to drive robot movements. Nowadays this science is becoming more and more important and integrated in our life. Applications began in the military field, as often happens with new technologies, but now mobile robotics is used also in industrial and domestic settings.

The work of this Master's Thesis is in the context of outdoor localization of autonomous vehicles. The aim of this project is to use raw GPS pseudorange data in the localization problem by creating the corresponding geometric constraint for each measurement. Thereafter the localization problem will try to satisfy these constraints, as well as other constraints coming from measurements provided by other sensors, to produce a vehicle pose estimate (position and orientation). This sensor fusion approach gives a degree of robustness to the final localization estimate.

This Master's Thesis has been carried out at the mobile robotics labora-

tory of the Institut de Robòtica i Informàtica industrial of Barcelona (IRI) [1], in the context of the European project Cargo-ANTS, devoted to investigate and develop novel techniques for autonomous navigation of wheeled vehicles in port terminal areas. This European project, introduced in the next section 1.2, represents a use case instance for the work developed through this Master's Thesis, which could be applied to other projects requiring robust outdoor localization of autonomous vehicles.

1.2 Cargo-ANTs Project

Cargo handling by Automated Next generation Transportation Systems, in short Cargo-ANTs [1], is an European project that aims to create smart Automated Guided Vehicles (AGVs) and Automated Trucks (ATs) that can cooperate to achieve an efficient and safe freight transportation in shared workspaces, such as ports and freight terminals.

Cargo-ANTs goals are to increase throughput of freight transportation, maintaining a high level of safety for workers and goods. Each smart vehicle must employ robust planning, decision, control and safety strategies.

To achieve these goals a reliable and accurate positioning system is required, as well as an environmental perception system that allows to detect moving and stationary objects, and to position them in a relative and absolute way. To coordinate the vehicle movements, dynamic path planning and docking point detection capabilities are also essential components.

Figure 1.1: Vehicles used to move containers in a freight terminal.

The project is funded by the European Community and involves five partners from three different countries: TNO and ICT Automatisering from

[1] http://www.iri.upc.edu/

Netherlands, Volvo Technology and Halmstad University from Sweden, and the Spanish National Research Council (CSIC), the largest public institution dedicated to research in Spain. Within Spain, research on this project is carried out at the Institut de Robòtica i Informàtica industrial of Barcelona (IRI), which is a Joint Research Institute that hosts researchers both from the Universitat Politècnica de Catalunya (UPC) and from the CSIC.

1.3 Structure of the Master's Thesis report

This document is structured as follows: in Chapter 2 we will present the SLAM problem, a key component in mobile robotics projects such as Cargo-ANTs. In the end of the same chapter we will introduce Wolf, the software framework that is being developed at IRI to solve SLAM.

In Chapter 3 we will overview the GPS, with a conceptual and mathematical explanation of how it works, and an overview on the communication between receivers and satellites.

The practical work done for this Master's Thesis is divided in two main parts. Firstly, Chapter 4 shows how raw GPS data is captured and processed in order to create useful constraints for solving a generic SLAM problem that fuses measurements from different kinds of sensors. In the end of the chapter it is also shown how to compute a GPS fix with these constraints, and there are some practical results from experiments with real data.

The second main part has been to integrate the raw GPS pseudorange constraints in Wolf, and to solve SLAM fusing GPS raw measurements with odometry. This will be shown in Chapter 5, followed by results from real experiments with a wheeled mobile robot.

In Chapter 6, conclusions of this Master's Thesis are discussed, with some possible future developments.

All the software developed in this Master's Thesis project is Open Source, and the code is hosted in different repositories, that are all listed in Appendix A.

2. Sensor Fusion in Mobile Robotics

2.1 Simultaneous Localization And Mapping

Simultaneous Localization And Mapping (SLAM) is the computational problem of incrementally constructing a consistent map of an unknown environment with a moving robot, while simultaneously determining its location within this map.

The SLAM problem is a key factor in all mobile robotics fields, in fact a solution to this problem would provide the means to make a robot truly autonomous [2].

The problem of robot localization in unknown environments is essential in any task that require motion, and the capability to build a map helps the robot to plan its movements according to the environment (obstacle avoidance, motion planning, robot learning etc.).

Nowadays SLAM is employed in domestic, industrial, health-care, military and security settings. Some examples are self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, human body inspection, until domestic tasks such as vacuuming or gardening.

When a mobile vehicle traverses an unknown environment, it measures its own movements, and detects external objects or features in this environment. These detections are called relative observations of landmarks. Using this data the vehicle builds a map and use it to get localized in it.

In SLAM both the trajectory of the platform and the location of all landmarks are estimated on-line without the need of any prior knowledge of location.

Figure 2.1 represents the essential SLAM problem: a mobile robot, represented as a triangle, moving through the environment using a sensor to take relative observations (red arrows) of landmarks (stars) [3, 4, 5].

At time k, the following quantities can be defined:

- $\boldsymbol{x}_k$: State vector that describe the pose of the robot. Pose means position and orientation.

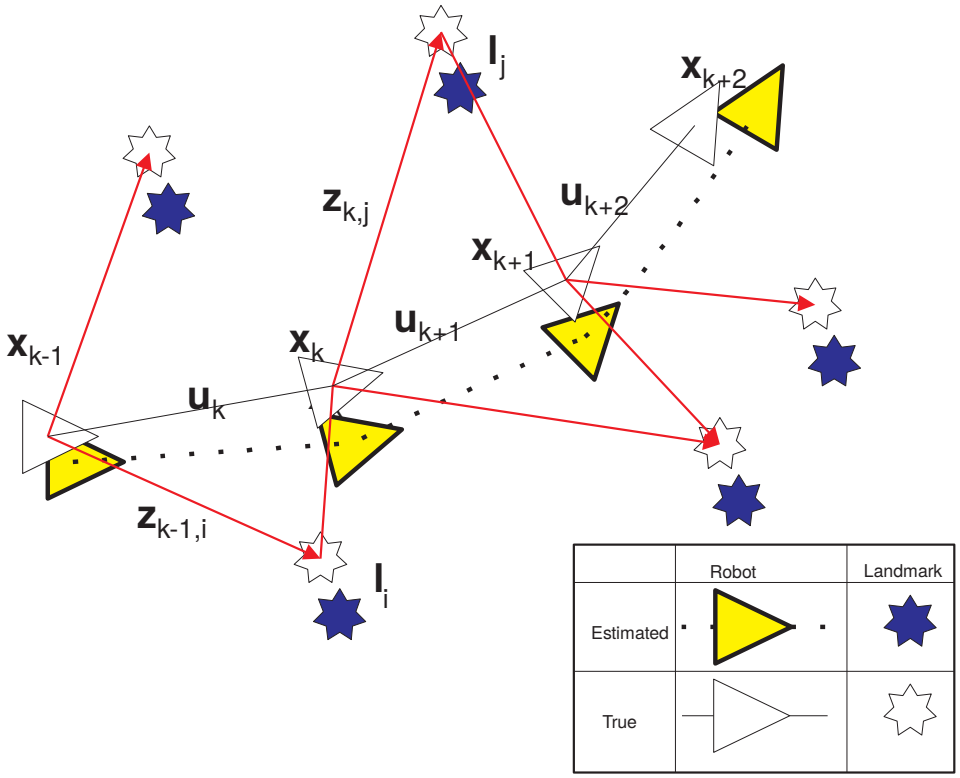


Figure 2.1: Simultaneous estimation of both robot trajectory and landmarks locations. It also shows how landmarks location is affected by the error in vehicle positioning.

- $\boldsymbol{l}_i$: A vector describing the location of the *i-th* landmark.
- $\boldsymbol{u}_k$: The control vector applied at time $k-1$ to drive the robot to the state $\boldsymbol{x}_k$ at time k from its previous state x_{k-1}.
- $\boldsymbol{z}_{ik}$: Observation of the *i-th* landmark at time k.

SLAM problem can be formalized in a probabilistic form [5, 6]. Each of the just defined quantities is a random variable, and they can be grouped in the following sets:

- $X = \{\boldsymbol{x}_0, \boldsymbol{x}_1 \ldots, \boldsymbol{x}_k\}$: The history of robot's poses. $X_{0..k}$ represents the robot's states from time 0 to time k, and it is also called trajectory of the robot.

- $L = \{\boldsymbol{l}_1, \boldsymbol{l}_2 \ldots, \boldsymbol{l}_n\}$: The sets of all landmarks (also known as map).
- $U = \{\boldsymbol{u}_1, \boldsymbol{u}_2 \ldots, \boldsymbol{x}_k\}$: The history of control inputs.
- $Z = \{\boldsymbol{z}_1, \boldsymbol{z}_2 \ldots, \boldsymbol{z}_k\}$: The sets of all landmark observations.

The pose of the robot at time k, $\boldsymbol{x}_k$, depends on the pose of the robot at the previous time, $\boldsymbol{x}_{k-1}$, and of the control $\boldsymbol{u}_k$ given at the robot.

$$P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{2.1}$$

An observation of a landmark, $\boldsymbol{z}_i$, depends on the pose the robot had when the measurement has been taken, $\boldsymbol{x}_k$, and on the pose of the observed landmark, $\boldsymbol{l}_i$.

$$P(\boldsymbol{z}_i \mid \boldsymbol{x}_k, \boldsymbol{l}_i) \tag{2.2}$$

The joint probability of trajectory, controls and measurements is the product of all these probabilities.

$$P(X, L, U, Z) \propto P(\boldsymbol{x}_0) \prod_k P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \prod_i P(\boldsymbol{z}_i \mid \boldsymbol{x}_k, \boldsymbol{l}_i) \tag{2.3}$$

A SLAM problem consists in maximizing the likelihood of the trajectory and landmark poses, so it means finding the sets X^* and L^* that maximize equation 2.3 which implies to find that state (poses and landmarks) that explains better all the measurements gathered during the movement.

$$\{X^*, L^*\} = \arg\max_{X,L} \left(P(\boldsymbol{x}_0) \prod_k P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \prod_i P(\boldsymbol{z}_i \mid \boldsymbol{x}_k, \boldsymbol{l}_i) \right) \tag{2.4}$$

These dependencies can be well represented by a factor graph (Figure 2.2). A factor graph is a bipartite graph representing the factorization of a function. In this graph there are two kind of vertices: variable vertices, that represent the state of the variables we want to estimate, and factor vertices, which are the constraints between the states.

Each state block is constrained to a small number of other blocks compared to the cardinality of the graph, so the resulting graph is called sparse. This sparsity is taken into account to solve the associated non-linear optimization problem with the most appropriate algorithms for this kind of problems, such as the incremental Cholesky factorization method [5].

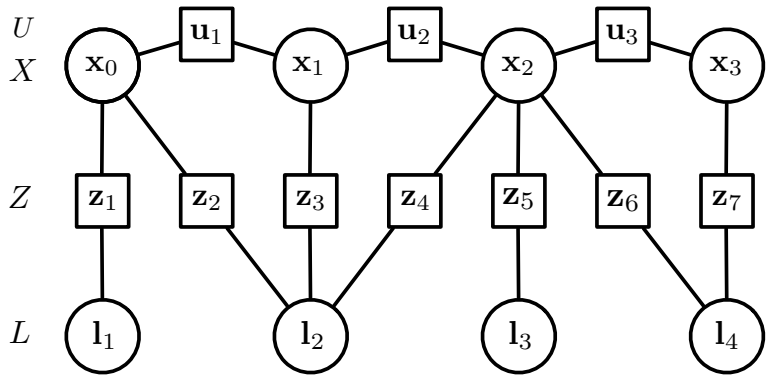


Figure 2.2: Factor graph of a SLAM problem [5]. Capital letters on the left of the graph indicate the set of random variables in which the nodes belong.

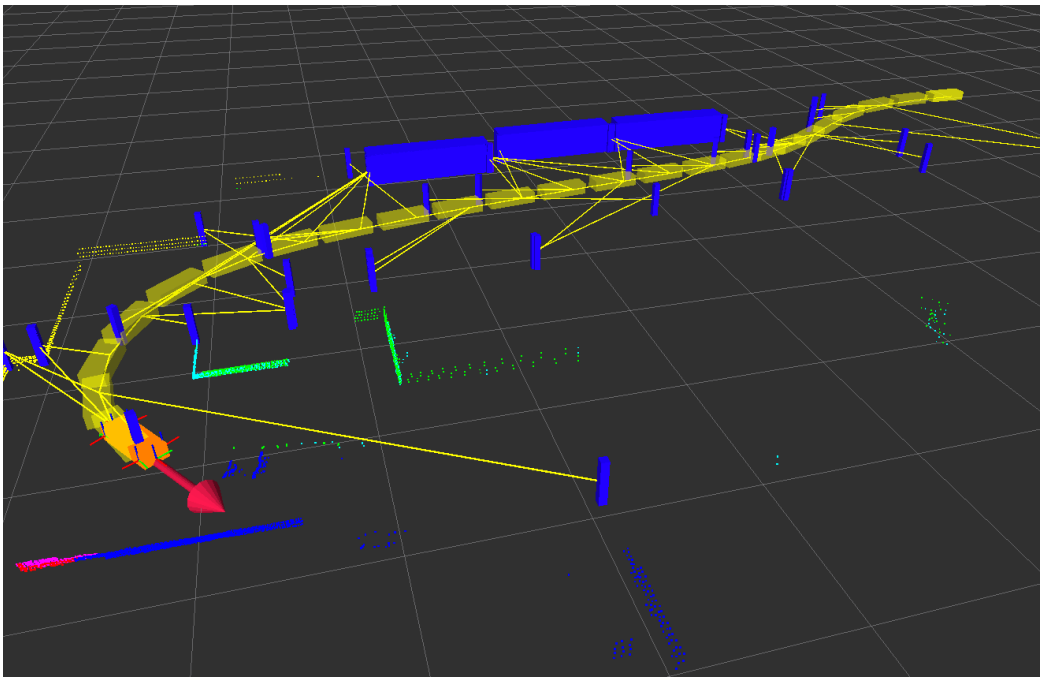


Figure 2.3: Example of SLAM in Cargo-ANTs. The vehicle, represented by a yellow rectangle, is moving in a freight terminal. During the time it builds a map, discovers landmarks (in blue) and creates constraints (yellow lines) between landmarks and the poses in its trajectory.

2.2 Sensor Fusion

When the robot moves in the map, it tries to localize itself in it observing landmarks and sensing its own motion. These observations, also called measurements, can be done with one sensor, but also with more sensors of the same kind, or with different types of sensors.

Sensor fusion means combining data from a set of heterogeneous or homogeneous sensors into a coherent and enhanced description of the surrounding environment.

A system that uses sensor fusion expects a series of advantages with respect to a single sensor system:

- Improved accuracy: When multiple independent measurements of the same property are fused, the accuracy of the resulting value is better than the one achieved with a single sensor.
- Robustness and reliability: Multiple sensor suites have an inherent redundancy which enables the system to provide information even in case of partial failure.
- Robustness against measurements outliers: By increasing the dimensionality of the measurement space the system becomes more robust to outliers (non coherent measurements).
- Reduced ambiguity and uncertainty: Joint information reduces the set of ambiguous interpretations of the measured value.
- Extended spatial and temporal coverage: One sensor can look where others cannot, or respectively can perform a measurement while others cannot.
- Increased confidence: A measurement of one sensor is confirmed by measurements of other sensors covering the same domain.

Sensor fusion is not a new concept, and theoretically exists from a long time, but nowadays the advances in sensor technology and processing techniques, combined with improved hardware, make real-time fusion of data possible in complex applications such as SLAM.

2.3 Windowed Localization Frames (WOLF)

The effort required to build and manage SLAM and sensor fusion is often huge. For that reason Wolf has been created. The acronym Wolf stands for Windowed Localization Frames and it is a versatile software framework for the localization of a mobile robot and mapping of the environment around it [7].

Wolf has been created in the IRI's group of mobile robotics, where it is currently being designed, improved and developed to offers solutions to a wide range of SLAM problems. Wolf development started at the same time of the Cargo-ANTs project, which is one of the possible applications. Anyway, Wolf is thought to be as general as possible, to address SLAM problem in many different contexts.

The philosophy behind Wolf is providing a framework that automates the basic functionality needed in a SLAM problem, freeing developers to focus on the engineering of localization solutions and not on the low level coding.

Wolf allows to configure a robotic system with virtually any number of sensors, of any kind, an make them work together.

The main structure in Wolf is a tree of base classes that describe the common elements of a SLAM problem (see Figure 2.4). The root of the tree is the *Wolf Problem*, from where three main branches start: *Hardware*, *Trajectory* and *Map*.

The *Hardware* branch contains a list of the utilized sensors with their parameters. The parameters that describe a property of the sensor, like time bias for an internal clock or focal length for a camera, are called intrinsic. Instead, the parameters that are not proper of the sensor, like the mounting 3D pose on-board a vehicle, are called extrinsic. The *Hardware* branch does not manage real sensor data acquisition, which is out of the the Wolf scope.

Each sensor is associated with one or more *Processor*, which is a class appointed to process the received raw data (called Capture, see below).

Another important branch is the *Map*. A map is composed by a set of *Landmarks*, that are features that describe the surrounding environment.

Every Wolf problem has a *Trajectory*, that is a list of reference *Frames* that define the state of the robot at a different time. Each frame has a list of *Captures*, that are objects where the raw data captured is stored.

The signal processor associated to the sensor that took a capture process the capture itself, and extract a set of metric measurements, called *Features*.

Each *Feature* leads to a list of *Constraints* for the optimization problem. With a *Constraint* it is possible to compute a residual, that is a discrepancy between the received measurement and the prediction of this measurement

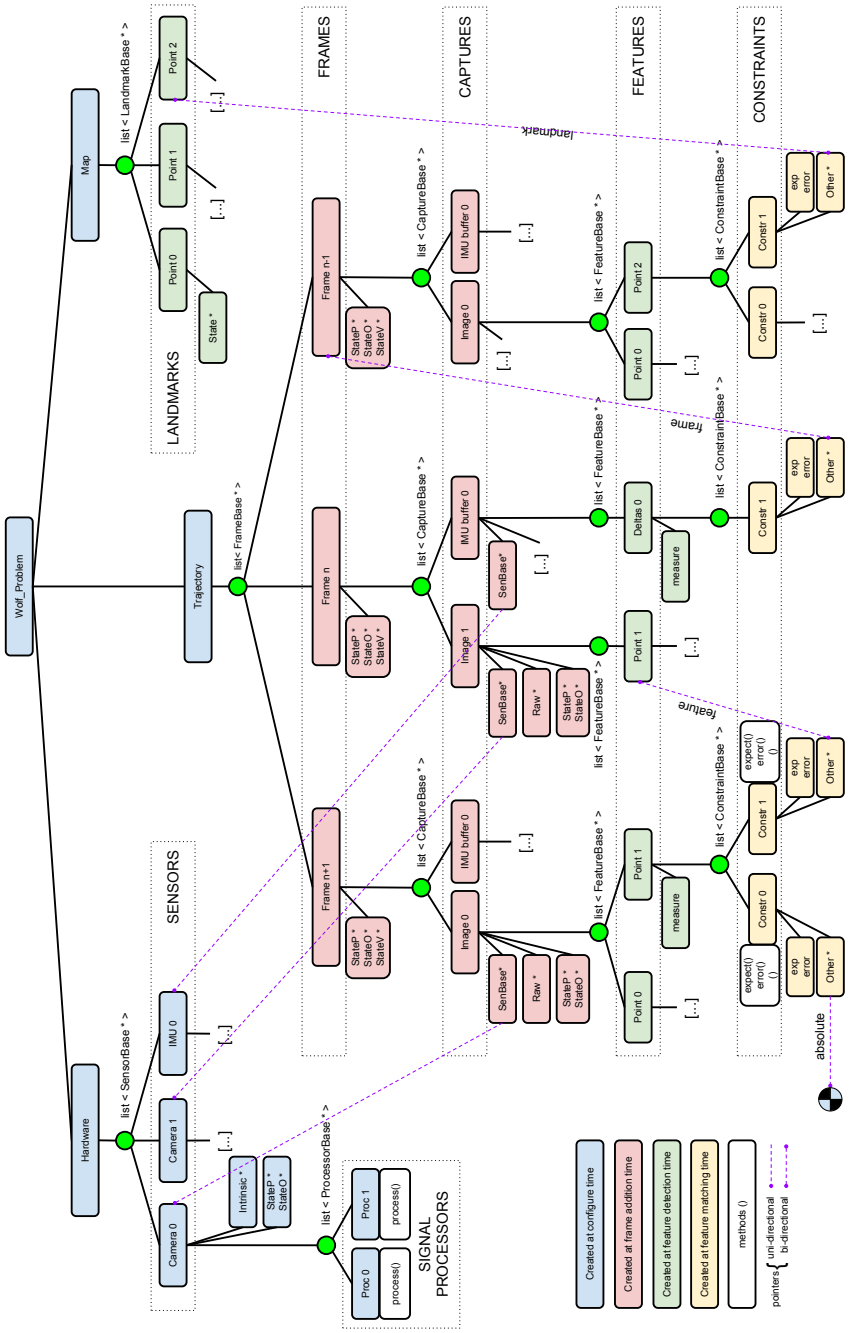


Figure 2.4: WOLF tree: a structure for storing all the information of the SLAM problem. Base classes are Sensor, Processor, Landmark, Capture, Feature and Constraint. For illustrative purposes, this diagram shows a case where the problem is solved with cameras and IMU, but Wolf is generic to which sensor modality is fused.

we can make based on the robot estimated state.

Wolf is sais windowed because the optimization problem can be restricted to only a subset of all the frames in the whole trajectory. The frames in this window, that are the ones that enter into the optimization problem, are called *Key-Frames*. By minimizing with a solver all the residuals associated to these key-frames we can compute the optimal state of the robot, that is the goal of the localization problem.

Wolf is mainly a structure to store data in an organized way. The task of solving an optimization problem is left to an external solver, that can be integrated with Wolf through a wrapper. Natively, Wolf provides a ready-to-use wrapper for Ceres-Solver [8], a solver created by Google that will be further described in section 4.1.

Chapter 5 of this report presents in detail how Wolf base classes have been specialized to solve the fusion problem of interest in this work, which is the fusion of an odometry source with raw GPS pseudorange measurements.

3. Global Positioning System (GPS)

3.1 GPS Fundamentals

The GPS is a Global Navigation Satellite System (GNSS) that provides worldwide accurate three-dimensional position, velocity and time information to users.

It was created, and it is maintained, by the United States government. Currently GPS and the Russian GLONASS (GLObal NAvigation Satellite System) are the only fully operational GNSS. European Union is developing another GNSS called Galileo and it is scheduled to be fully operational by 2020.

GPS is a dual-use system: it provides separate services for civil and military users. These are called the Standard Positioning Service (SPS) and the Precise Positioning Service (PPS) respectively. The SPS is designated for the civil community, whereas the PPS is intended for U.S. authorized military and select government agency users. Civilian use is permitted, but only with special U.S. Department of Defense approval.

The SPS is available to all users worldwide free of direct charges. There are no restrictions on SPS usage: it is freely accessible to anyone with a GPS receiver. This service is specified to provide accuracies of better than 13m (95%) in the horizontal plane and 22m (95%) in the vertical plane (global average; signal-in-space errors only). However, SPS measured performance is typically much better than specification, [9].

In order to provide this information, the GPS system leans against a constellation of satellites, also called Space Vehicles (SVs). The satellite constellation nominally consists of 24 satellites arranged in 6 orbital planes with 4 satellites per plane. A worldwide ground control/monitoring network monitors the health and status of the satellites. This network also uploads navigation and other data to the satellites, [10, p.382].

GPS is available everywhere on the Earth and in all weather condition, provided that there is a line of sight to four or more GPS satellites in order to calculate a precise 3D position. There are techniques to obtain a position

in situation where less satellites are available to the receiver, and we touch upon later on that aspects. GPS can provide service to an unlimited number of users since the user receivers operate passively.

The GPS concept is based on time. The satellites carry very stable atomic clocks that are synchronized to a GPS time base. The synchronism between all the satellites' clock with a unique time base is maintained from ground clocks, and any drift from GPS time base is corrected daily from them. GPS receivers have clocks as well. Usually a crystal clock is employed to minimize the cost, complexity, and size of the receiver, so it is less accurate than the satellites' clock.

GPS satellites continuously broadcast their current time and navigation data. The navigation data provides the means for the receiver to determine the location of the satellite at the time of signal transmission, whereas the ranging code enables the user's receiver to determine the transit time of the signal and thereby determine the satellite-to-user range.

A GPS receiver monitors multiple satellites and from them creates and solves a system of equations to determine the exact three-dimensional user's position. If the receiver's clock were synchronized with the satellites' clock, only three range measurements would be required. But, because of the lower precision of crystal clocks, the receiver is not synchronized with true GPS time. Thus, four satellites must be in line of sight with the receiver for computing the user's three-dimensional position and receiver's clock bias from GPS time. If either GPS time bias or height is accurately known, less than four satellites are required.

The three-dimensional position is calculated in Cartesian coordinates with origin in the Earth's center. The reference coordinate system chosen to represent both the satellite and the receiver is the *Earth-Centered Earth-Fixed Coordinate System* (ECEF). The receiver's Earth-centered solution location is usually converted to geodetic coordinates.

The peculiarity of ECEF is that the x-, y-, and z-axes rotate with the Earth and no longer describe fixed directions in inertial space. The +x-axis points in the direction of 0° longitude and 0° latitude, the +y-axis points in the direction of 90°E longitude and 0° latitude, and the z-axis is chosen to be normal to the equatorial plane in the direction of the geographical North Pole, thereby completing the right-handed coordinate system.

3.2 Position Determination

3.2.1 Concepts of Trilateration

To solve the user's position a trilateration technique is used, and it requires the user-to-satellite distances to be measured. These measurements are computed using time information.

The satellites embed in their signals the time of transmission. That enables the receivers to calculate the time of flight subtracting the time of transmission from the time of arrival.

Multiplying the signal's time of flight by the speed of light, the pseudorange measurement is obtained.

With only one satellite's measurement, the user would be located somewhere on the surface of a sphere centered on the satellite, as shown in Figure 3.1

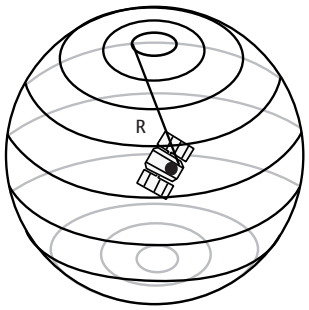


Figure 3.1: The user is located on the surface of the sphere with center on the satellite position and radius equal to the pseudorange.

Using simultaneously the pseudorange of a second satellite, another sphere of possible results is found. The intersection of the two spheres shrinks the set of possible user positions to the perimeter of the shaded circle in Figure 3.2, that denotes the plane of intersection of these spheres, or at a single point tangent to both spheres (i.e., where the spheres just touch). This latter case is highly unlikely and could only occur if the user were collinear with the satellites, which is not the typical case on the Earth surface.

The plane of intersection is perpendicular to a line connecting the satellites, as shown in Figure 3.3

Adding a third satellite to this measurement process permits to complete the trilateration and determine the user position. The set of possible posi-

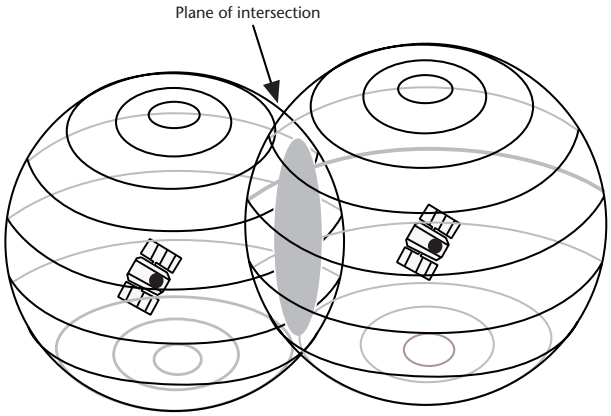


Figure 3.2: The user is located somewhere on the perimeter of the shaded circle.

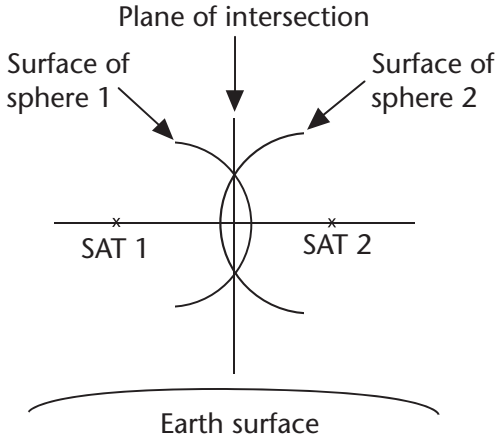


Figure 3.3: Plane of intersection between the two spheres.

tions is now shrunk to the intersection of the perimeter of the circle and the surface of the third sphere.

With the intersection between the perimeter of the circle represented in Figure 3.4 as shaded, and the surface of the third sphere, we obtain two points. However, only one of this two points is the correct user position. With the visual aid of the intersection given by Figure 3.5 we can observe that the candidate locations are mirror images of one another with respect to the plane generated by the 3 involved satellites.

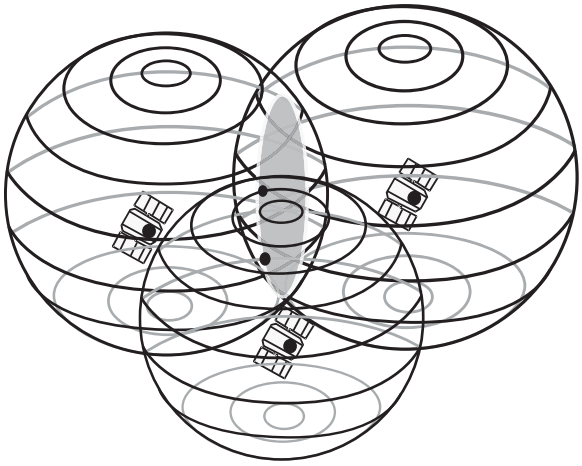


Figure 3.4: The user is in one of the two points on the shaded circle.

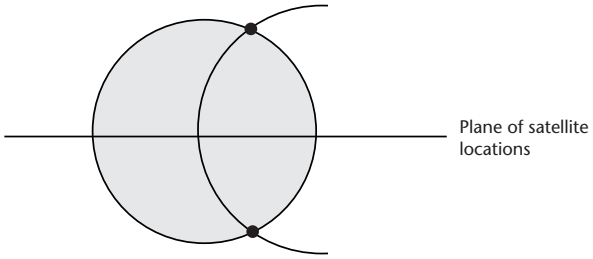


Figure 3.5: The user is in one of the two points on circle perimeter.

Assuming that the user is on the Earth's surface we can immediately discard the upper point and use the lower one as the true position of the receiver.

For spaceborne receiver the trilateration process is more complex, because they may be above or below the plane containing the satellites, and it may not be clear which point to select unless the user has supplementary information, such as tracking [11, pp. 21–26]. Anyway it is out of the scope of this thesis and will not be investigated.

In all of these examples the receiver clock is assumed perfectly synchronized to system time. With real GPS receiver this assumption never holds, because there are a number of error sources that affect range measurement accuracy (e.g., measurement noise and propagation delays). These can generally be considered negligible when compared to the errors experienced from

non-synchronized clocks. Therefore, in the explanation of basic concepts, errors other than clock offset are omitted.

3.2.2 Mathematical Formulation

In order to deepen the analysis of position determination into mathematical terms, we can make reference to Figure 3.6.

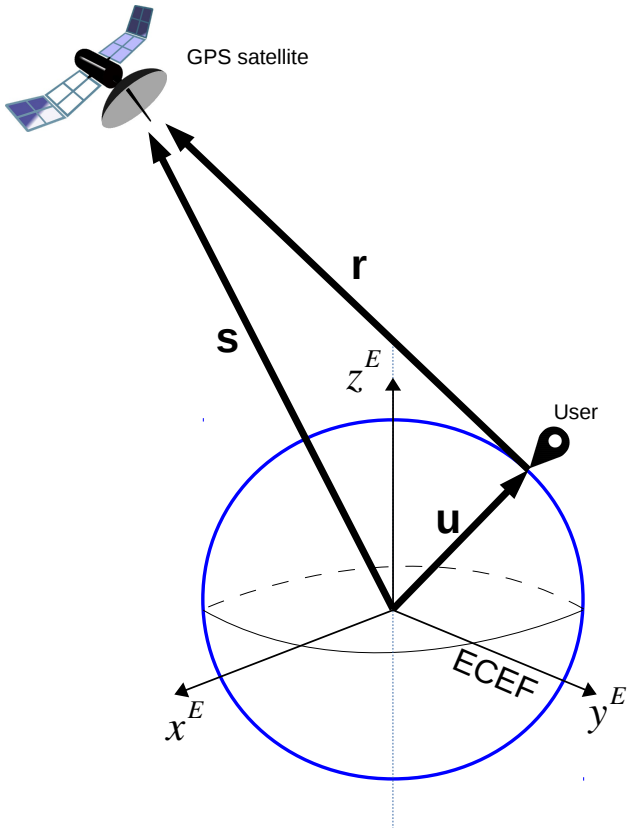


Figure 3.6: Relations between Earth, user and GPS satellite represented through vectors.

Our goal is to determine the vector **u**, which represents a user receiver position with respect to the ECEF coordinate system origin. Thus, the user position coordinates x_u, y_u, z_u are considered unknowns.

Vector **s** represents the position of the satellite relative to the coordinate origin, and the coordinates x_s, y_s, z_s are computed using navigation data broadcasted by the satellite.

Vector **r** represents the vector offset from the user to the satellite and can be calculated as

$$\boldsymbol{r} = \boldsymbol{s} - \boldsymbol{u} \tag{3.1}$$

Let r represent the magnitude of vector **r**.

$$r = \|\boldsymbol{r}\| = \|\boldsymbol{s} - \boldsymbol{u}\| \tag{3.2}$$

The user-to-satellite distance r is computed by measuring the propagation time required by the signal sent from the satellite to reach the receiver. The propagation time is represented by Δt and in the ideal case is:

$$\Delta t_{ideal} = t_{Rx} - t_{Tx} \tag{3.3}$$

If the satellite clock and the receiver clock were perfectly synchronized, the correlation process would yield the true propagation time. By multiplying this propagation time, Δt, by the speed of light, the true (i.e., geometric) satellite-to-user distance could be computed. This is the ideal case as described in Section 3.2.1, but, as stated before, the satellite and receiver clocks are generally not synchronized. The receiver clock will usually have a bias error from system time that we will call t_b, which is unknown. Taking into consideration this quantity we obtain the propagation time in the non-ideal case:

$$\Delta t = t_{Rx} - t_{Tx} + t_b \tag{3.4}$$

Thus, the range determined by the correlation process is denoted as the pseudorange ρ, because it is the range determined by multiplying the signal propagation velocity, c, by the time difference between two non-synchronized clocks (the satellite clock and the receiver clock).

This ρ, calculated as $\rho = \Delta t \cdot c$, is considered as the *measurement* received from the satellite.

Calculation of User Position

In order to determine user position in three dimensions (x_u, y_u, z_u) and the time bias t_b, pseudorange measurements ρ_j are required from four satellites resulting in the next system of equations, where j references each current satellite.

Given the user position $\boldsymbol{u}$, the receiver bias t_b and the j-th satellite position $\boldsymbol{s}_j$, the expected pseudorange $\hat{\rho}_j$ is

$$\hat{\rho}_j = \|\boldsymbol{s}_j - \boldsymbol{u}\| + c \cdot t_b \tag{3.5}$$

This equation can be expanded in x_u, y_u, z_u and t_b:

$$\hat{\rho}_j = \sqrt{(x_{s_j} - x_u)^2 + (y_{s_j} - y_u)^2 + (z_{s_j} - z_u)^2} + c \cdot t_b \tag{3.6}$$

After that we can compute the error as the difference between received and expected measurement, ρ_j and $\hat{\rho}_j$:

$$e_j = \rho_j - \hat{\rho}_j \qquad j \in \{Currently\ tracked\ satellites\} \tag{3.7}$$

Note that in this equation the unknowns are four: the user position x_u, y_u, z_u and the time bias t_b.

To find the optimal result for this four unknowns, we have to minimize the sum of squared errors corresponding to the satellites observed in that epoch.

$$\begin{aligned}(x_u, y_u, z_u, t_b)^* &= \underset{(x_u, y_u, z_u, t_b)}{\arg\min} \sum_j (\hat{\rho}_j - \rho_j)^2 \\ &= \underset{(x_u, y_u, z_u, t_b)}{\arg\min} \sum_j \left(\sqrt{(x_j - x_u)^2 + (y_j - y_u)^2 + (z_j - z_u)^2} + c \cdot t - \rho_j \right)^2 \end{aligned} \tag{3.8}$$

where x_j, y_j, z_j denote the j-th satellite's position in three dimension.

When more than four satellites are available, we can choose if using only the four more reliable measurements or use more than four simultaneously.

3.3 GPS Signals

In the previous section we assumed to know satellites' position and pseudoranges every time we want to trilaterate the receiver position. In order to provide these information satellites broadcast two different signals, called respectively observation and navigation data.

3.3.1 Observation Data

Observation data contains a ranging code that enables the user's receiver to determine the propagation time of the signal, and thereby determine the satellite-to-user *pseudorange*. In order to do that, GPS observables include three fundamental quantities that need to be defined:

- *Time*: The time of the measurement is the receiver's time of the received signals. It is identical for all the satellites observed in a observation, and it is also called *epoch*. It is expressed with respect to GPS time and not to Coordinated Universal Time UTC.

- *Pseudorange*: The pseudorange is the measured distance from the receiver antenna to the satellite antenna, including receiver clock bias and other error sources such as atmospheric delays. It is measured comparing two clocks but it is converted and stored in units of meters, multiplying the time measurement by the speed of light.

- *Phase*: The phase is the carrier-phase measured in whole cycles at the frequencies used to transit the signal. The original civil signal is transmitted using two different bands: 1575.42 MHz called L1 and 1227.60 MHz called L2 [12]. Anyway, not all the receivers track both signals.

Observation data messages are kept very small for the sake of being computed at a high rate by the receiver.

3.3.2 Navigation Message

In addition to the pseudoranges, a receiver needs to know detailed information about each satellite's position and the full GPS constellation. This information is sent to receivers from satellites through the navigation message, and allows the users to compute the satellite's orbit and the precise position at each moment, to perform the positioning calculus when observation message arrives.

The navigation message includes:

- *Time Parameters and Clock Corrections*: to compute satellite clock bias and time conversions.

- *Ephemeris Parameters*: to precisely estimate the satellite's orbit during time and compute the satellite coordinates with enough accuracy. Each satellite transmits its own ephemeris.

- *Service Parameters*: that contain satellite health information.

- *Ionospheric Parameters Model*: to correct the ionospheric effect on single frequency observations.

- *Almanacs*: permit the computation of the position of the satellites constellation with the reduced accuracy of 1-2 km of 1-sigma error, which assists the receiver in determining which satellites search for. Each satellite transmits almanac data for several satellites.

In order to being able to describe almanacs and to maintain a highly accurate time synchronization between the space vehicles, every satellite exchanges information with fixed ground stations on Earth's surface. GPS ephemeris message includes not only parameters to calculate satellites position, but also the time of their applicability. The ephemerides and clock parameters are usually updated every two hours, but anyway they are considered still valid for four hours. Instead the almanac is valid for six days. [13]

Navigation data is heavier than observation data, and it changes every two hours. For these reasons it has been chosen to send this data in a different message, separated from observables, and at the considerably slower rate of one every thirty seconds. Thus, when a GPS sensor is turned on, it have to wait maximum thirty seconds in order to be able to start trilaterating the user position, if at least four satellites are visible.

The whole navigation message consists of 25 frames of 30 seconds each. A frame is 1500 bit long, and is subdivided in 5 sub-frames of 300 bits, numbered 1 to 5. In turn each sub-frame consists of 10 words of 30 bit each and requires 6 seconds to transmit (see Figure 3.7). Each sub-frame has the GPS time. The first sub-frame contains the GPS week number and information to correct the satellite's time to GPS time, plus satellite status and health. Sub-frames 2 and 3 together contain the transmitting satellite's ephemeris data. Sub-frames 4 and 5 contain only 1/25th of the complete almanac. A receiver must process 25 whole frames to retrieve the entire 15,000 bit almanac message. At this rate, 12.5 minutes are required to receive the entire almanac from a single satellite.

A short discussion is needed to justify the ionospheric parameters. The signals from the GPS satellites are perturbed as they transit the ionosphere. Pseudorange measurements are increased in value because an additional delay is added to the time of flight. For this reason a correction is needed, otherwise the positioning would not be accurate.

If the receiver works with both frequencies $L1 = 1575.42MHz$ and $L2 = 1277.60MHz$, a linear combination of the measurements removes almost all of the ionospheric perturbations [14]. Lets call ρ^{L1} and ρ^{L2} the pseudoranges measured respectively on $L1$ and $L2$, it is possible to correct the measurements applying a ionospheric correction $\Delta_{ionospheric}$ in this way:

$$\tau = \left(\frac{L1}{L2}\right)^2 \tag{3.9}$$

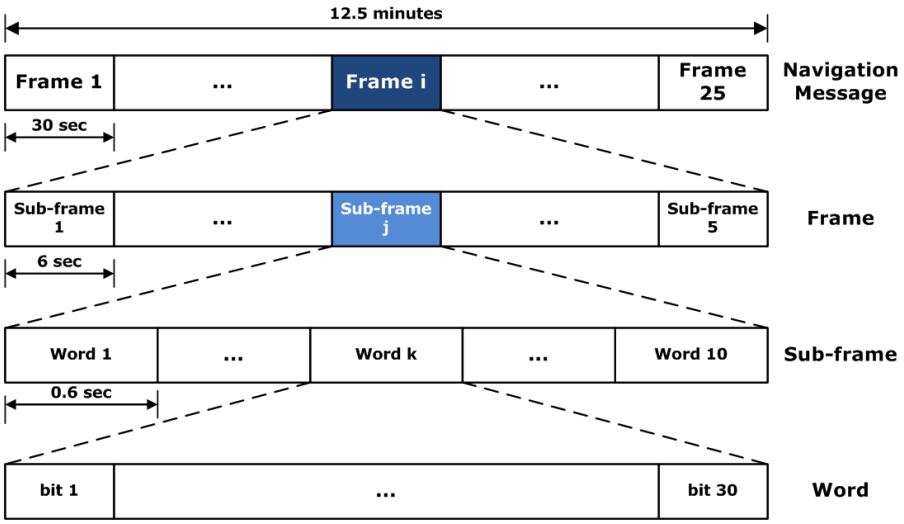


Figure 3.7: Structure of the navigation message.

$$\Delta_{ionospheric} = \frac{\rho^{L1} - \rho^{L2}}{1 - \tau} \tag{3.10}$$

$$\rho_{corrected} = \rho^{L1} - \Delta_{ionospheric} \tag{3.11}$$

If the receiver works with only one frequency it is relevant to apply a precise ionospheric model. This model can be built from the parameters included in the navigation message.

The navigation message format described previously is called L1 C/A navigation message, and represents the legacy message. Nowadays the system has been modernized and new type of messages has been introduced: L2-CNAV, CNAV-2, L5-CNAV and MNAV. Anyway, in modernized GPS, the legacy navigation message is still used, but transmitted at a higher rate and with improved robustness, and the new messages contains additional data for civil or military users [13].

4. Software Development I. GPS Data Processing

In this chapter we will present the implementation of the satellites' orbit estimation and the trilateration of the user position. The programming language chosen for this project is C++, for performance reason and because of compatibility with other tools that we will use. One of these tools is ROS, and now we are going to introduce it. This will help the reader to understand the architecture of this project solution.

ROS (Robot Operating System) [15] is a framework for robot software development. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Its libraries are in C++ and Python, but also other languages are supported. ROS is an open source software released under the terms of the BSD license. It was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory, and now is maintained by the Open Source Robotics Foundation.

A key concept in ROS are the nodes. A node is a process that performs some computation. A node can publish or subscribe to data streams called topics, which allow communication between them. A robot application usually is composed by many nodes, each one responsible of a different aspect. The use of nodes in ROS provides several benefits to the overall system, like improving fault tolerance and diminishing the code complexity.

One relevant node provided by ROS is RVIZ. This node is specialized in visualization. It can visualize various data format that are published from other nodes through topics, such as coordinates frames and visual markers to visualize the robot and the surrounding environment.

In order to have a general overview of the contents that are presented later on in this chapter, in Figure 4.1 we illustrate the final architecture, contained in the package `raw_gps_ros`, and in the following we will briefly describe the goal of each node.

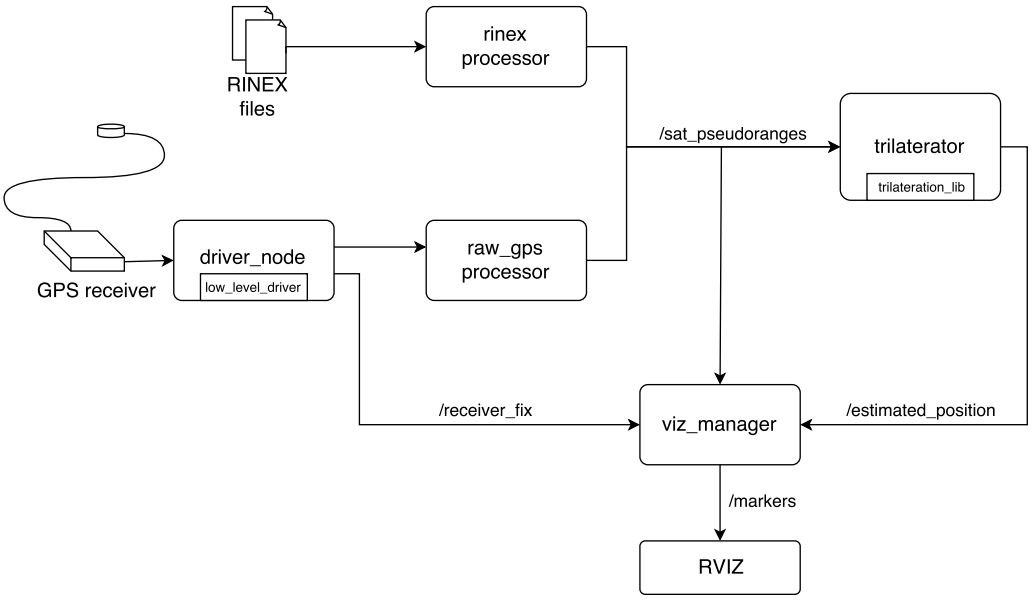


Figure 4.1: Architecture of the `raw_gps_ros` package. The arrows without names represent sets of relations, that are detailed in Figures 4.3 and 4.6.

`raw_gps_ros` can use alternatively two kinds of inputs. The first option is to read the raw GPS data stored in specific datasets, called RINEX files (see section 4.2). The second option is to capture it directly through a real GPS receiver, that is controlled from `driver_node` (see section 4.3).

The raw GPS data is processed respectively from `rinex_processor` and `raw_gps_processor`. These nodes publish in the `/sat_pseudoranges` topic a vector of satellite positions and their corresponding pseudoranges for every epoch.

On the top right we can see the `trilaterator` node. Its task is to listen to the `/sat_pseudoranges` topic and, every time some pseudoranges are published, starting to trilaterate the user position. As soon as it computes a position and a clock bias, it publishes the result in the `/estimated_position` topic.

Another node, called `viz_manager`, listens to all the topics and produces the visualization objects, that will be visualized through the `rviz` node.

4.1 Trilateration with Ceres-Solver

To trilaterate the user position as described in section 3.2.2 we have to optimize the non-linear least squares problem described in equation 3.8. To solve this problem efficiently we use a specific solver called Ceres-Solver.

Ceres-Solver is a library for modeling and solving large, complicated optimization problems [8]. It is an open source C++ library created by Google in 2010 in order to estimate the pose of Street View cars and aircrafts. Ceres is still used by Google and other important companies and during its life span it has been extensively tested and optimized. The code is portable and can run on different operating systems, like Linux, Windows, Mac OS X, Android and iOS.

Ceres accepts non-linear least squares problems with bounds constraints, structured like

$$\min_{\mathbf{x}} \frac{1}{2} \sum_i \xi_i \left(\| f_i \left(x_{i_1}, ..., x_{i_k} \right) \|^2 \right) \qquad l_j \leq x_j \leq u_j \tag{4.1}$$

where $\xi_i \left(\| f_i \left(x_{i_1}, ..., x_{i_k} \right) \|^2 \right)$ is called `ResidualBlock` and is composed with the `CostFunction` $f_i(\cdot)$ and the `LossFunction` ξ_i. The `LossFunction` is used to reduce the influence of the outliers on the solution of non-linear least squares problems.

As we saw in section 3.2.2, the positioning problem is exactly in the same form of ours (see equation 3.8), with:

$$\texttt{LossFunction} \quad \xi_i \equiv 2 \tag{4.2}$$

$$\texttt{CostFunction} \ f_i(\cdot) = \sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} + c \cdot t - \rho_j \tag{4.3}$$

The first step to find an optimized solution through Ceres is to write a cost functor class, that will be used by the solver to evaluate our formula.

The core of this class is the `operator()` method, in which the error is estimated subtracting the real measurement from the expected measurement, and saved in `residual[0]`. It is important to notice that `operator()` is a templated method, and `T` can be a double or a special type called `Jet` that let us use the automatic differentiation offered by Ceres, without having to write manually the Jacobian computation code.

```
class CostFunctor
{
public:
 CostFunctor(const SatelliteMeasurement sm_, const double
    speed_) : sm(sm_), speed(speed_)
 {}

 template <typename T>
 bool operator()(const T* const pos, const T* const bias, T*
    residual) const
 {
   T square_sum = T(0);

   for (int i = 0; i < sm.pos.coords.size(); ++i)
     square_sum += pow(pos[i] - T(sm.pos.coords[i]) , 2);

   T distance = (square_sum != T(0)) ? sqrt(square_sum) : T(0);
   residual[0] = (distance + bias[0]*speed) - sm.pseudorange;
   return true;
 }

private:
 SatelliteMeasurement sm;
 const double speed;
};
```

When we want to trilaterate a position we create a `ResidualBlock` based on this `CostFunction` class for each satellite visible in that moment, and then we ask the optimizer to try to find the solution that minimizes the sum of errors of all pseudoranges currently measured.

To test the trilateration through Ceres, we have created a library called *Trilateration*. To trilaterate a position are needed at least four satellites and the pseudoranges between the receiver and them. To specify that data it is possible to directly insert in the command line each satellite's position $\boldsymbol{s}$ and pseudorange ρ. Instead of specifying the pseudoranges, it is possible to insert the real receiver position $\boldsymbol{u}$, the clock bias t_b and a value for the standard deviation σ of the noise. If the second input type is chosen, the pseudoranges ρ_j will be simulated considering the true range $r_j = \|\boldsymbol{r}_j\|$ between the j-th satellite and the real receiver, the clock bias t_b and a Gaussian white noise $\mathcal{N}(0, \sigma^2)$ will be applied.

$$\rho_j = r_j + c \cdot (t_b + t_n) \; ; \qquad t_n \sim \mathcal{N}(0, \sigma^2) \tag{4.4}$$

After this step the pseudoranges will be available, and it is possible to start to build the problem and then optimize. A `ResidualBlock` is added

to the problem for each satellite. The minimization behaviour chosen to be used from the solver is the *Trust Region*. This approach approximates the objective function using a quadratic model function over a subset of the search space, called trust region. If the model function applied lower the cost of the objective function the trust region expanded, otherwise is contracted.

To help the convergence of the solution it is possible to provide an initial guess of the solution, and this leads to more fast convergence and accurate results.

In Figure 4.2 we can see the output screen of an execution of the trilateration. At the beginning it simulates some pseudorange measurements, knowing that the real receiver's position is $\boldsymbol{u} = (200, 200, 20)$ and clock bias $t_b = 2.5ms$, and applying a white Gaussian noise $\mathcal{N}(0, \sigma^2)$ with standard deviation $\sigma = 5 \cdot 10^{-9}$ to the clock bias. After that Ceres begins the optimization, starting from an initial guess of $\boldsymbol{u} = (0, 0, 0)$ for the receiver's position and $1\mu s$ of clock bias. After twelve iterations the result converges, leading to a position that is $0.57m$ far from the real position. The estimated clock bias is almost perfect, given that the original one is $2.5ms$ and the estimated one is only $1.07ns$ more.

```
Measure 0: [(-2500, 3060, 14000), 764002.614191]
Measure 1: [(1000, -3200, 15000), 764861.434245]
Measure 2: [(-6000, -800, 17000), 767586.575239]
Measure 3: [(2400, 5500, 15600), 766084.464992]
Measure 4: [(9900, 1600, 18800), 770665.955871]
Measure 5: [(36543, 19845, 46512), 811675.361282]
Measure 6: [(-84954, -65498, 65432), 875362.937787]

iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  1.965910e+12    0.00e+00    1.57e+15   0.00e+00   0.00e+00  1.00e+04        0    2.65e-04    4.00e-04
   1  8.114054e+05    1.97e+12    2.12e+10   3.93e+03   1.00e+00  3.00e+04        1    2.81e-04    7.45e-04
   2  2.193456e+04    7.89e+05    1.50e+11   3.96e+03   9.75e-01  9.00e+04        1    2.15e-04    9.75e-04
   3  6.144913e+00    2.19e+04    1.44e+09   1.77e+02   1.00e+00  2.70e+05        1    2.05e-04    1.20e-03
   4  4.126534e+00    2.02e+00    2.56e+04   2.15e+00   1.00e+00  8.10e+05        1    2.28e-04    1.46e-03
   5  4.126534e+00    4.09e-08    1.50e+00   8.78e-04   9.98e-01  2.43e+06        1    2.36e-04    1.72e-03
   6  4.126534e+00    6.88e-11    6.98e-02   2.17e-07   2.72e+04  7.29e+06        1    2.43e-04    1.99e-03
   7  4.126534e+00   -2.25e-11    0.00e+00   2.91e-10  -2.41e+09  3.64e+06        1    2.45e-05    2.04e-03
   8  4.126534e+00   -2.25e-11    0.00e+00   2.91e-10  -2.41e+09  9.11e+05        1    2.06e-05    2.08e-03
   9  4.126534e+00   -2.25e-11    0.00e+00   2.91e-10  -2.41e+09  1.14e+05        1    1.08e-05    2.12e-03
  10  4.126534e+00   -2.25e-11    0.00e+00   2.91e-10  -2.41e+09  7.12e+03        1    1.05e-05    2.14e-03
  11  4.126534e+00    1.89e-11    1.05e-01   2.89e-10   2.03e+09  2.14e+04        1    2.06e-04    2.36e-03
Ceres Solver Report: Iterations: 11, Initial cost: 1.965910e+12, Final cost: 4.126534e+00, Termination: CONVERGENCE

Initial receiver guess: [(0, 0, 0), 1e-06]      |       noise std dev = 5e-09

Real receiver:          [(200, 200, 20), 0.0025]
Estimated receiver:     [(200.179533706, 200.196754852, 20.5047713704), 0.00250000106766]

Coords difference:      (-0.179533705814, -0.196754852329, -0.504771370398)
Coords distance:        0.570735                        (0=good)
Bias ratio:             1                       (1=good)
```

Figure 4.2: Output screen of a test of the trilateration library.

4.2 Compute Satellite Position from Ephemeris

A fundamental step in the user positioning process is to calculate the position of the satellites in a certain moment, usually when a GPS observation has been sent. As described in subsection 3.3.2, the information needed to compute this data is sent in the GPS navigation messages. To calculate the orbit of a single satellite we use a C++ library called GPSTk.

GPSTk (GPS Toolkit) [16] is an open source C++ library that provides a wide range of functions that solve processing problems associated with GNSS, such as manage and convert time representations, ephemeris calculations, compute a position solution and processing standard formats like RINEX. The GPSTk is developed by Space and Geophysics Laboratory, within the Applied Research Laboratories at the University of Texas at Austin.

With GPSTk we can compute the satellites' position in a certain moment of the day, given that we know the corresponding ephemerides. In this library ephemerides are stored in objects of the `GPSEphemeris` class. Each object contains all the parameters needed to compute the evolution of a single satellite in its orbit.

In the following of this section we will discuss about how to fill these objects, because it can be done in different ways and needs a longer discussion.

From each ephemeris it is possible to compute the corresponding satellite position at the given time through the method `GPSEphemeris::svXvt(time)`. This method implements the equations of orbital motion as defined in IS-GPS-200 [17]. It returns an `Xvt` object that contains the satellite's Earth-Centered, Earth-Fixed Cartesian position, velocity, clock bias and drift. From now on we will refer to this Position, Velocity and Time object as PVT.

Before explaining how to fill the ephemerides, it is necessary to introduce the RINEX file format.

RINEX (Receiver Independent Exchange Format) [18] is a data interchange format for raw satellite navigation system data. RINEX can store data from different GNSS, like GPS (including GPS modernization signals e.g. L5 and L2C), GLONASS, Galileo, Beidou, etc. The RINEX standard allows to store data in a format independent from the receiver used. This allows the user to easily exchange the receiver device and also to post-process it with other data unknown to the original receiver, like using an accurate atmospheric model.

Three different RINEX file types exist: Observation data file, Navigation message file and Meteorological data file. Each file type consists of a header

section and a data section. The format has been optimized for minimum space requirements and there is no maximum record length limitation for the observation records.

Observation data file contains a list of observation messages as described in section 3.3.1.

Navigation message file contains a list of all the epochs in which observations are made and the corresponding ephemerides. In order to keep the file small, ephemerides are saved only when a new satellite is seen or when they are no more valid, and not every thirty seconds as they are sent by satellites.

Meteorological data file contains information about the weather. They can not be produced by the receiver itself, but they must be provided by an external source. The use of this data is out of the scope of this project.

We can fill an ephemeris reading from a RINEX navigation file or receiving data directly from a GPS sensor. With GPSTk it is straightforward to read ephemerides contained in a RINEX nav file, by means of using the class `Rinex3NavStream` and extract all the ephemerides through the overloaded `operator<<()`. Instead, if we want to use a real device, we need a GPS sensor capable to provide the raw data received from the satellites, and not only the GPS fix. In this work we used a Septentrio AsteRx1 GPS/Galileo Single-frequency receiver. In section 4.3 we will discuss about the custom driver needed to publish the raw data and fill a `GPSEphemeris` object.

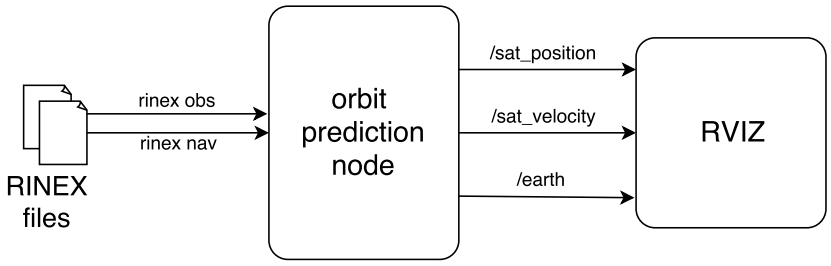


Figure 4.3: Architecture of `orbit_estimation` node that reads data from RINEX files.

In our representation through the ROS node RVIZ there is a frame called world, that represent the center of the Earth in the ECEF coordinates. Around this point we can see a green sphere that represents the Earth, to better visualize the satellites' evolution. In Figure 4.4 we can see eight satellites, each of them in its own frame, moving around the Earth. The light blue arrows represent the velocity of each space vehicle. The red line we can see behind one of the satellites is its orbit. It doesn't appear circular only

because the ECEF coordinate system is not inertial, since it moves with the Earth rotation.

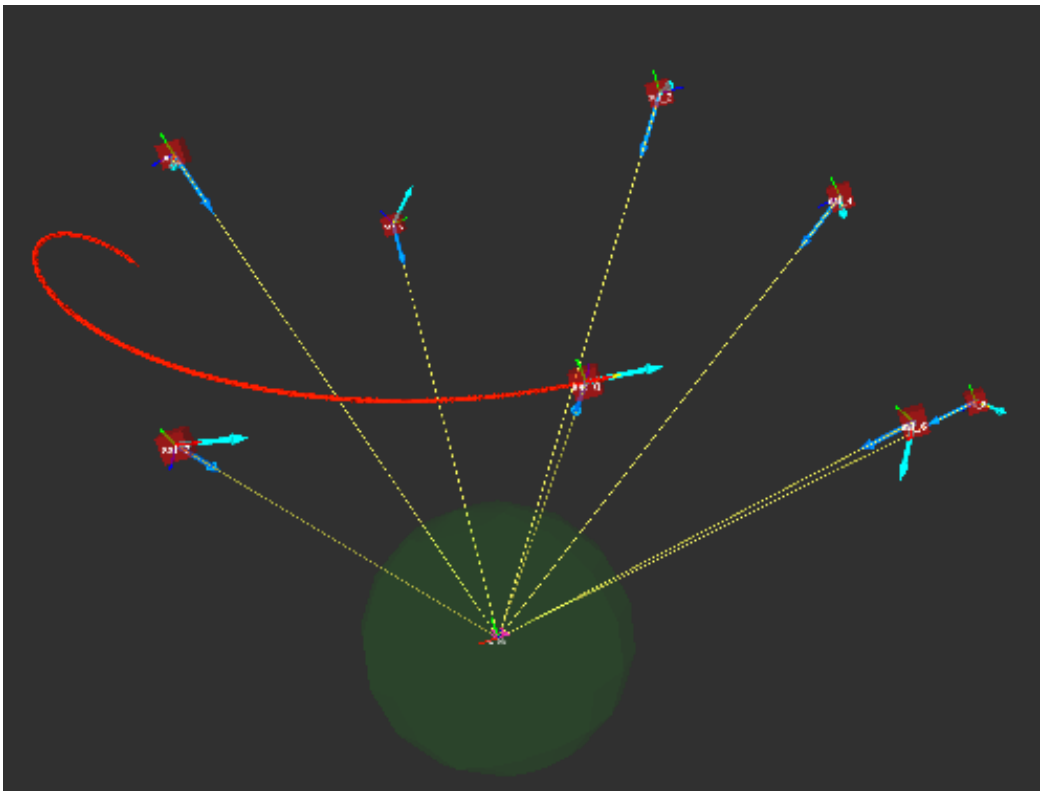


Figure 4.4: Representation on RVIZ of the satellite positioning around the Earth.

4.3 Custom Driver for Septentrio AsteRx1

The device we used to get the real GPS data is a Septentrio AsteRx1 GPS/-Galileo single-frequency receiver. It is composed by a compact OEM board with low power consumption integrated in the receiver platform and an active high-gain antenna. These components can be seen in Figure 4.5, the receiver platform on the left and the external antenna on the right, respectively. The antenna is external so it can be placed in the best position to guarantee visibility of satellites. The platform provides an USB interface that allows to connect AsteRx1 to a computer. Once connected, the receiver can be configured through a command line interface [19], and it is compatible both with Linux and Windows operating systems.

Figure 4.5: Septentrio AsteRx1 GPS/Galileo single-frequency receiver. On the left there is the receiver device and on the right the external antenna.

The binary output format of Septentrio receivers is SBF, that straightforwardly comes from Septentrio Binary Format [20]. In this format, data is arranged in binary blocks. Copious different SBF blocks exist, but we are interested only in few of them.

This receiver contains 24 hardware channels for simultaneous tracking all the visible GPS and Galileo satellites. Each visible satellite in a certain moment is automatically allocated in a channel.

For each measurement epoch, AsteRx1 creates an output SBF block called `MeasEpoch` that contains a measurement for each tracked satellite. All the measurement set of a single `MeasEpoch` block refers to the same time.

The reference time in AsteRx1 is kept as close as possible to the currently used GNSS time, in our case GPS time. Anyway, as explained in subsection 3.2.2, the receiver's clock irremediably contains a time bias with respect to the GPS time. AsteRx1's time is stored in the SBF block `TimeStamp` and

it is composed by two numbers: the time-of-week `TOW` and the week number counter `WNc`. `TOW` is expressed in milliseconds and refers to the milliseconds elapsed from the beginning of the current week, and the `WNc` refers to the number of complete weeks elapsed since January 6, 1980.

The GPS fix produced by AsteRx1 is contained in `PVTCartesian` and `PVTGeodetic` blocks, that respectively refer to ECEF coordinate system and latitude, longitude and altitude in the Geodetic coordinate system.

Anyway, for the scope of our project, we want to focus on the raw data sent by GPS satellites and not on the fix produced by the receiver.

In order to get the raw data from AsteRx1, a custom driver was needed. This driver uses the command line interface provided by Septentrio in order to retrieve the desired data, and offers an API to manage the receiver from outside [1]. We will refer to this part of the driver as the *low-level driver* for the GPS receiver.

To use the GPS receiver in our architecture, a ROS node has been created. This node wraps the low-level driver and, using the provided API, receives data from AsteRx1 and publishes the information we need in different topics. In Figure 4.6 we can see the driver node, called `driver_node`, and its most significant topics. The `/gps` and `/gps_ecef` topics contain a ROS message with the GPS fix computed by AsteRx1, respectively in Geodetic coordinates and ECEF. `/gps_meas` contains the observation data described in subsection 3.3.1.

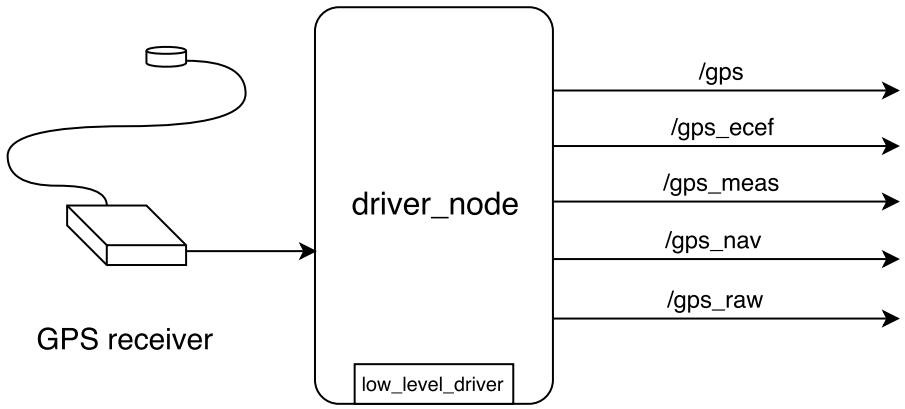


Figure 4.6: Architecture of the driver node for the GPS receiver AsteRx1.

[1] The code of this low-level driver can be found at `https://devel.iri.upc.edu/labrobotica/drivers/asterx1_gps`

To compute the satellite position in a certain moment, the corresponding ephemeris must be known. AsteRx1 can publish a GPS decoded message with clock and ephemeris data through the SBF block `GPSNav`. This block contains more than forty parameters needed to reconstruct the ephemeris. The ROS node `driver_node` publishes this set of parameters in the `/gps_nav` topic. As we wrote in the previous section, another node will listen to this topic in order to build the GPSTk's `GPSEphemeris` object. To construct it, all the parameters contained in the `GPSNav` block must be assigned to the `GPSEphemeris` object.

After some weeks of attempts without good result, we found out that GPSTk permits to construct another ephemeris object, `EngEphemeris`, using the first three sub-frames of the navigation data (see 3.3.2). For this reason we changed the low-level driver in order to publish the raw navigation data in the `/gps_raw` topic.

Each sub-frame is composed by ten words of 30 bits each. To construct the `EngEphemeris` we need to instantiate an empty object and call the method `addSubframe(..)` for three times to add the sub-frames. After that it is possible to convert the `EngEphemeris` into a `GPSEphemeris` and use it in the same way as one read from a RINEX navigation file.

4.4 Computing GPS Fix from Pseudoranges

Now that we know how to trilaterate with Ceres and how to estimate the satellite's position in a certain moment, we can compute a GPS fix from the data received from satellites. In order to do it, firstly it is necessary to compute the satellite positions from ephemerides, and then to apply a correction to the received pseudoranges. Now, if we have four or more satellite we can prepare the Ceres-problem as described in the section 4.1 and then find an optimized solution. and then it is possible to trilaterate. In the following of the section we will explore these steps.

4.4.1 Correct Raw Pseudorange Measurements

Every time we receive a GPS observation, that means a vector of satellites and their respective pseudoranges, we have to check if we have already received the corresponding ephemeris in order to calculate the satellite's position. If the correct ephemeris is unknown we cannot use that satellite, and we simply discard it.

Anyway, this process is not as straightforward as it seems, because we need the satellite position at the transmit time t_{Tx}, and with the GPS obser-

vation received we only have the receiving time t_{Rx}. To calculate it precisely, we have a two-step process: we firstly calculate a raw estimation of the transmit time t^1_{Tx}:

$$t^1_{Tx} = t_{Rx} - \frac{\rho_{raw}}{c} \tag{4.5}$$

Now we need to get the PVT of the satellite in that moment:

$$PVT^1 = svXvt(t^1_{Tx}) \tag{4.6}$$

From this first guess of PVT, PVT^1, we can obtain the satellite's clock bias and calculate the relativistic clock correction. The rate of advance of two identical clocks, placed respectively one on the terrestrial surface and the other in the satellite, will differ due to the difference of the gravitational potential (general relativity) and to the relative speed between them (special relativity) [11, p. 306].

The relativistic clock correction $t^{PVT^1}_{\gamma}$ can be calculated as

$$t^{PVT^1}_{\gamma} = -2 \cdot \frac{\mathbf{s}^{PVT^1} \cdot \mathbf{v}^{PVT^1}}{c^2} \tag{4.7}$$

where $\mathbf{s}^{PVT^1} = PVT^1.pos$ and $\mathbf{v}^{PVT^1} = PVT^1.vel$ refer respectively to the satellite position (m) and velocity (m/s) vectors.

Using the satellite's clock bias $t^{PVT^1}_b = PVT^1.clockBias$ we can calculate the true transmit time t_{Tx}:

$$t_{Tx} = t^1_{Tx} - t^{PVT^1}_b - t^{PVT^1}_{\gamma} \tag{4.8}$$

and finally the satellite PVT at that time:

$$PVT = svXvt(t_{Tx}) \tag{4.9}$$

From the PVT we can correct the raw pseudorange received considering the satellite's clock bias and the relativistic clock correction.

$$\rho_{corrected} = \rho_{raw} + c \cdot (t^{PVT}_b + t^{PVT}_{\gamma}) \tag{4.10}$$

4.4.2 Receiver Autonomous Integrity Monitoring

Now that we have calculated the corrected pseudorange we could directly publish the satellite positions and pseudoranges and let the trilateration node work. Anyway, if we have at least five satellites in line of sight, we can apply a Receiver Autonomous Integrity Monitoring (RAIM) algorithm.

RAIM is a technique that uses an overdetermined solution to perform a consistency check on the satellite measurements. The RAIM algorithm compares the smoothed pseudorange measurements among themselves to ensure that they are all consistent. A pseudorange that differs significantly from the expected value may indicate a fault of the associated satellite or another signal integrity problem usually caused by ionospheric dispersion [11, p. 347].

If an outlier is found, it can be detected and excluded. GPSTk offers an implementation of this algorithm in `PRsolution2`.

RAIM algorithm is used by both our GPS data processors nodes and the GPS receiver AsteRx1 [21].

4.4.3 Trilaterate from Real Data

Pseudoranges correction and RAIM technique are implemented in the ROS nodes `rinex_processor` and `raw_gps_processor`. They act as receivers, with the only difference that one reads the raw data from RINEX files and the other one receives it from the driver node as described in section 4.3. When they receive a GPS measurement they calculate the satellite position in ECEF coordinates, correct the pseudoranges and eventually apply RAIM algorithm. In the end they publish a vector of satellite measurements in a topic called `/sat_pseudoranges` and they keep listening for new observations (see Figure 4.1).

One of the nodes that subscribes the `/sat_pseudoranges` topic is the node `trilaterator`. It uses the trilateration library described in section 4.1 to estimate the receiver's position and bias and publish this results to `viz_manager` node, that will visualize all the data it received through RVIZ.

4.5 Trilateration Results

In this section we analyze the estimation of the user position done by the `trilaterator` node, using the raw pseudorange measurements and ephemerides processed by the processor nodes previously described. Figure 4.7 shows the results obtained, with a blue dot for every GPS fix computed [2].

To compare the results obtained, in Figure 4.8 are shown the GPS fixes produced directly by the receiver, doing the same path. As we can see, the red path is much more neat compared to the blue one, but the shape of the entire path is correct.

[2]Figures 4.7 and 4.8 are created using an on-line utility called GPS Visualizer `http://www.gpsvisualizer.com/`

Figure 4.7: GPS fixes produced from raw GPS measurements by `raw_gps_ros`'s nodes.

Figure 4.8: GPS fixes produced directly by AsteRx1 receiver.

To produce such a smooth path AsteRx1 receiver uses a series of techniques to filter results [21]. One of these is the dynamic of the receiver, that is set to pedestrian. This means that the receiver for every new estimation will check the distance traveled, using the position of the last known fix as a reference. If the distance between these two spots is not compatible with the distance a pedestrian can cover in the same time, the estimation will be considered faulty.

Furthermore, AsteRx1 takes advantages of GNSS augmentation such as Satellite-Based Augmentation System (SBAS) [11, 22] to increase accuracy of the satellite positioning.

The only outlier rejection technique we are applying is the RAIM algorithm described in section 4.4.2. Analyzing the behaviour of the obtained results we noticed that our solution, but also the GPS fix produced by the receiver itself, are very sensible to multipath interference, and this is the main cause of outliers.

We created a three dimensional representation of the two path overlapping, using the `viz_manager` node and `RVIZ`. In Figure 4.9 we can see the two paths overlapped, to better understand the differences. On two dimensions the estimated positions are close to the GPS fix, with some estimate that

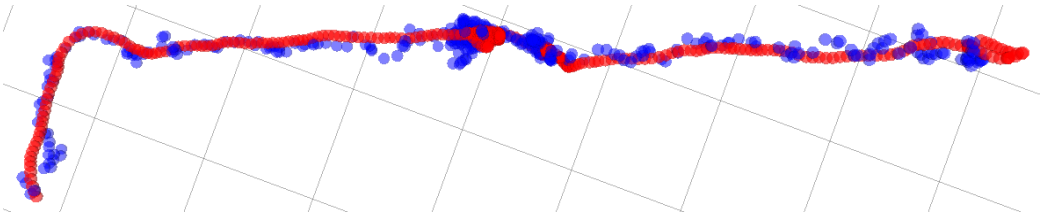


Figure 4.9: Three-dimensional representation of the two paths. In blue it is represented the path estimated by our implementation, in red the path estimated by the receiver. The size of the image is proportional to the grid, whose cells are squares with sides of 10 meters.

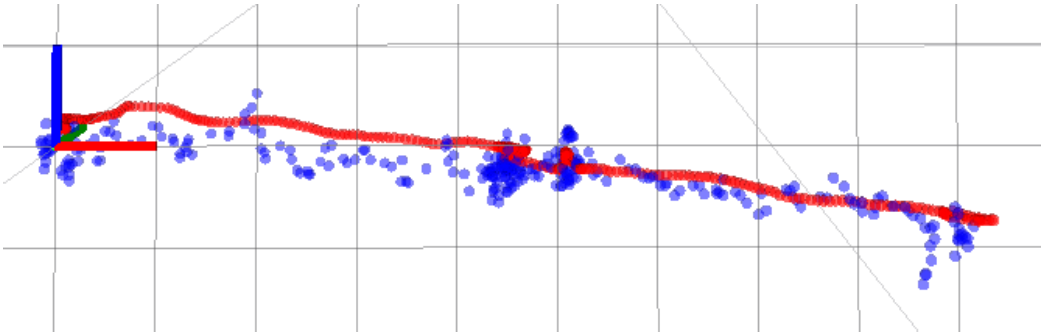


Figure 4.10: Lateral view of the estimated paths, to show the altitude flaw. The size of the image is proportional to the grid, which lies on the XZ plane, and whose cells are squares with sides of 5 meters.

distance itself a pair of meters.

The least accurate estimation regards the altitude, as we can see from Figure 4.10. Altitude is always the flaw of the GPS positioning because of the geometrical structure of the problem. To overcome to this weak point, the software of commercial GPS receivers reference altitude measurements to the altitude of the ground in the geodetic model of the Earth and on the last known position [23].

5. Software Development II. Wolf Specialization

5.1 Wolf Integration

The second main step of the work for this thesis has been to integrate the GPS functionalities in Wolf (see section 2.3). To do that, it has been necessary to specialize most of the base classes present in the Wolf tree.

A `SensorGPS` class has been created to represent the GPS sensor, from the point of view of the SLAM problem. In this class there are all the parameters needed to describe a GPS sensor. Among them, there is the sensor position, that represents the mounting point of the GPS antenna with respect to the robot. The only intrinsic parameter of a GPS receiver is its clock bias with respect to the GPS time. Two other values saved in this class are the position and orientation of the *Map* frame. The concept of frame is very important, and in the next section will be exhaustively explained.

With the GPS sensor, it is associated a signal processor, implemented in the `ProcessorGPS` class. This class is mainly composed by a method to process a GPS capture, extract the features from there and create from each feature a constraint for the optimization problem. In the GPS case, a `CaptureGPS` corresponds to a GPS observation with a vector of pseudoranges and satellite positions, as published from the `raw_gps_processor` or the `rinex_processor` nodes. That means that every satellite position and pseudorange received are already corrected, as explained in subsection 4.4.1.

In the GPS case, we chose to create a different feature for every received pseudorange, so the class `FeatureGPSPseudorange` contains only a three-dimensional vector for the satellite position and a real number for the pseudorange. To each feature it is associated a single constraint. The class `ConstraintGPSPseudorange2D` is the most complex class of the specialized one for the GPS, as it is the templated class that have to work with Ceres-Solver. Basically, the operations that must be done to compute the residual are the same implemented in the `trilateration` library and described in

section 4.1. The small but substantial difference is that we don't know and don't want to estimate the sensor position in the ECEF coordinates, but we want to localize the robot with respect to the Map.

In the next section we will introduce the concept of reference frame and how a set of these frames is structured in our work. After that we will show the important difference between the simple `operator()` of the implementation in the node described in the chapter 4 and the `operator()` in Wolf.

5.2 Reference Frames Structure

An essential concept in mobile robotics is the notion of reference frame. A reference frame consists of an abstract coordinate system and it is uniquely fixed in the space by the position of its origin of coordinates and its orientation with respect to its *parent frame*. The parent of all the frames is called *global frame* and it is considered static and fixed somewhere in the space.

The position $\boldsymbol{p}_B^F$ and orientation $\boldsymbol{\Phi}_B^F$ of a rigid body B relatively to the frame F together are called *pose* of B in F, which is noted as $B^F = (\boldsymbol{p}_B^F, \boldsymbol{\Phi}_B^F)$.

In the three-dimensional space, a position $\boldsymbol{p}$ is a 3D point and an orientation $\boldsymbol{\Phi}$ can be represented with a quaternion or with a 3x3 rotation matrix.

In our problem, we can identify four different frames. The global frame is called *ECEF* and corresponds to the ECEF coordinate system. A robot moves into a limited area of the world. This area is represented through a frame named *Map*, and it is a child of the *ECEF* frame, as we can see in Figure 5.1.

This transforms the relative positioning of the robot with respect to the Map to an absolute positioning on Earth, in ECEF coordinates.

Inside this Map we can see the robot's frame, called *Base* frame, that represents the robot position and orientation with respect to the Map. In turn, *Base* frame is the reference frame of all the sensor mounted in the robot, and in our case we have another frame called *GPS*. In the GPS case particularly, the orientation of this frame is not important, because a GPS antenna is omnidirectional.

If Figure 5.2 we can see in detail the Map and the trajectory the robot had respect to *Map* frame.

Now we can see the substantial difference between this situation and the simpler problem of localizing the GPS antenna directly in the ECEF coordinates. In this latter case we know the sensor position with respect to the *Base* frame, and we have to express it in the *ECEF* frame.

Formalizing mathematically this problem, we can call the GPS sensor position with respect to the *Base* frame $\boldsymbol{S}^B$, and we want to calculate it with

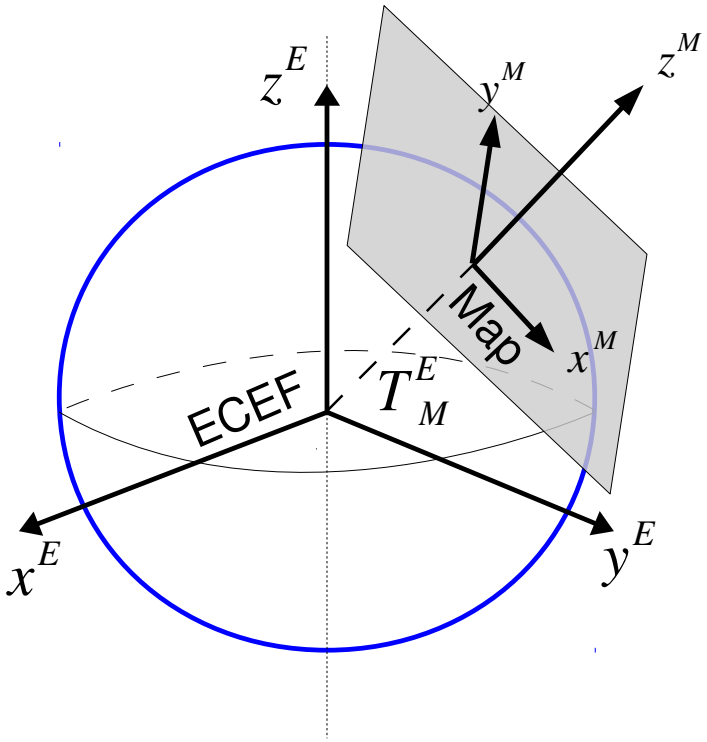


Figure 5.1: *Map* and *ECEF* frames. T_M^E transforms the relative positioning of the robot with respect to the Map to an absolute positioning on Earth, in ECEF coordinates.

respect to the *ECEF* frame, $\boldsymbol{S}^E$.

This frame transformation can be done using quaternions or homogeneous transformation matrices, and we chose the latter option. Such homogeneous transformation matrix for 3D points is a 4x4 matrix composed like this:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$$

where $\boldsymbol{R}$ is the rotation matrix and $\boldsymbol{t}$ the translation of the origin. Note that a generic 3D point $\boldsymbol{x}$ is padded with a 1, so it is represented as $\boldsymbol{x} = (x\ y\ z\ 1)^T$.

This is in order to work with the 4x4 transformation matrices with both rotation and translation.

Going back to our problem, we need to find a transformation matrix that transforms a point from *Base* frame to *ECEF* frame, that is noted as $\boldsymbol{T}_B^E$.

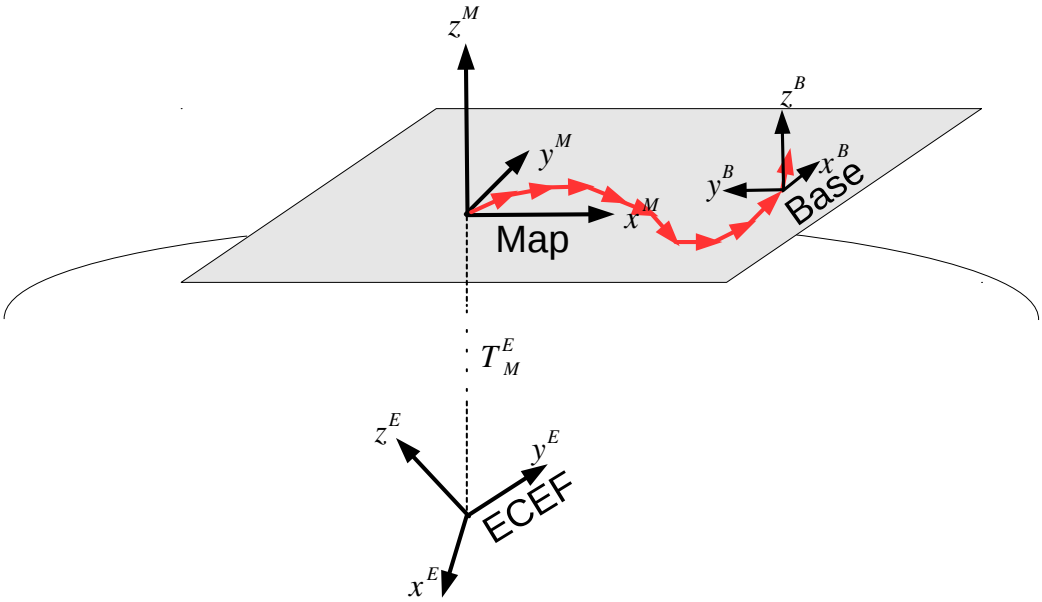


Figure 5.2: Trajectory of the robot inside the Map.

Knowing that matrix, we can transform the point between frames through

$$\boldsymbol{S}^E = \boldsymbol{T}_B^E \cdot \boldsymbol{S}^B \tag{5.1}$$

This transformation can be split into two different partial transformations:

$$\boldsymbol{T}_B^E = \boldsymbol{T}_M^E \cdot \boldsymbol{T}_B^M \tag{5.2}$$

We first calculate the sensor respect to *Map* frame $\boldsymbol{S}^M = \boldsymbol{T}_B^M \cdot \boldsymbol{S}^B$ and then we transform it in the *ECEF* frame with $\boldsymbol{T}_M^E$.

Lets use Figure 5.3 to help us analyze the first transformation $\boldsymbol{T}_B^M$. In this figure we are in the plane defined by the axes $\boldsymbol{x}^M$ and $\boldsymbol{y}^M$. The *Base* frame is translated $\boldsymbol{b}^M$ from the Map origin and rotated θ^M, so the transformation matrix $\boldsymbol{T}_B^M$ will be:

$$\boldsymbol{T}_B^M = \begin{bmatrix} \cos(\theta^M) & -\sin(\theta^M) & 0 & b_x^M \\ \sin(\theta^M) & \cos(\theta^M) & 0 & b_y^M \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

The second transformation, $\boldsymbol{T}_M^E$, is conceptually similar, but it involves a

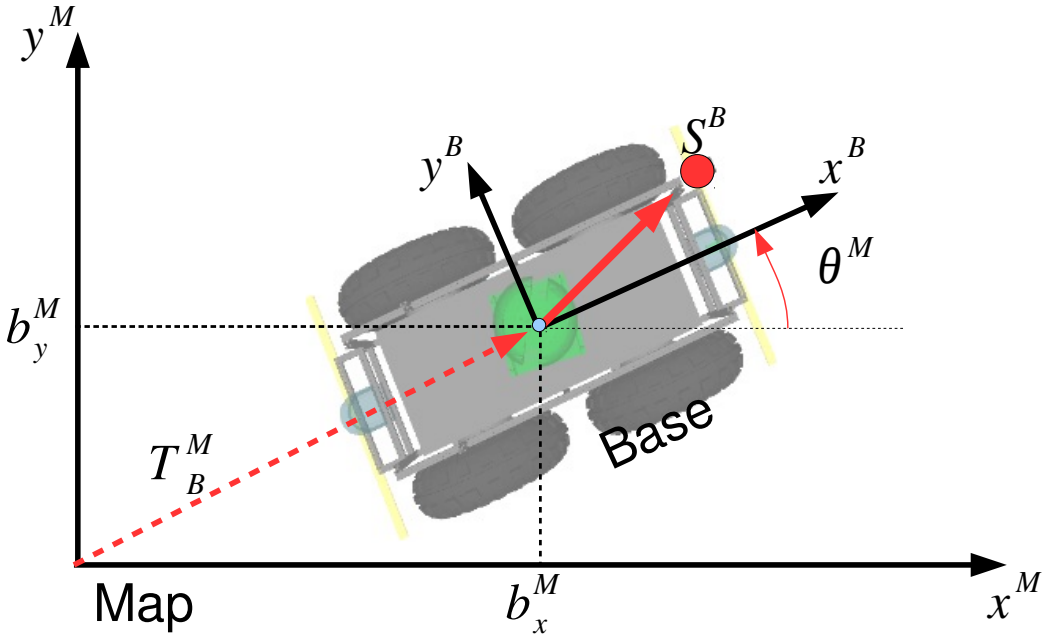


Figure 5.3: Representation in 2D of the robot's frame *Base* in the *Map* frame.

conversion between ECEF and geodetic coordinates and five different frame transformations.

First of all lets see how to convert $\boldsymbol{m}^E$, the origin of the frame *Map*, from ECEF to Geodetic. We use WGS84 as a reference ellipsoid to describe the Earth, which define the following constants:

$$\text{semi-major axis:} \quad a = 6378137 \tag{5.4a}$$

$$\text{first eccentricity:} \quad e = 8.1819190842622 \cdot 10^{-2} \tag{5.4b}$$

$$\text{semi-minor axis:} \quad b = \sqrt{a^2 \cdot (1 - e^2)} \tag{5.4c}$$

$$\text{second eccentricity:} \quad e' = \sqrt{\frac{a^2 - b^2}{b^2}} \tag{5.4d}$$

Then we define the following auxiliary values:

$$p = \sqrt{(m_x^E)^2 + (m_y^E)^2} \tag{5.5a}$$

$$\psi = atan\frac{a \cdot m_z^E}{b \cdot p} \tag{5.5b}$$

and finally we calculate the Geodetic coordinates of the Map origin:

$$\text{longitude:} \quad \phi = atan\frac{m_y^E}{m_x^E} \tag{5.6a}$$

$$\text{latitude:} \quad \lambda = atan\frac{m_z^E + e'^2 \cdot b \cdot \sin(\psi)^3}{p - e^2 \cdot a \cdot \cos(\psi)^3} \tag{5.6b}$$

$$\text{radius of curvature:} \quad N = \frac{a}{\sqrt{1 - e^2 \cdot \sin(\phi)^2}} \tag{5.6c}$$

$$\text{altitude:} \quad h = \frac{p}{\cos\phi} - N \tag{5.6d}$$

Remembering that $\boldsymbol{T}_M^E$ means representing the pose of *Map* with respect to *ECEF*, we can describe visually how we compute this computation. The visual description is followed by the associated homogeneous transform matrix.

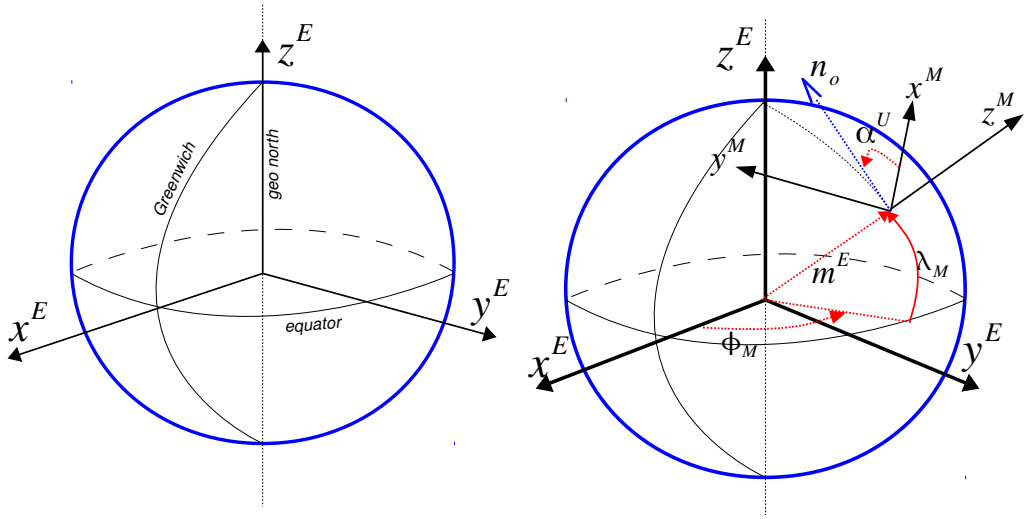


Figure 5.4: Geoid and relations between *ECEF* and *Map* frames [24]

The first thing we have to do is to translate the frame from the center of the Earth to the origin of the frame *Map*.

$$\boldsymbol{T}_A^E = \begin{bmatrix} 1 & 0 & 0 & m_x^E \\ 0 & 1 & 0 & m_y^E \\ 0 & 0 & 1 & m_z^E \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

After that we have to rotate the frame on the z-axis of an angle equivalent to the longitude ϕ of the frame *Map*,

$$\boldsymbol{T}_{\phi}^{A} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

and then rotate it on the y-axis of an angle equivalent to latitude λ.

$$\boldsymbol{T}_{\lambda}^{\phi} = \begin{bmatrix} \cos(\lambda) & 0 & -\sin(\lambda) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\lambda) & 0 & \cos(\lambda) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

Now we want to exchange the axes in order to transform to East-North-Up (ENU) coordinate systems, that means to having the x-axis pointing to the East, the y-axis to the North, and the z-axis normal to the ground.

$$\boldsymbol{T}_{U}^{\lambda} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.10}$$

Finally it is necessary to apply a last rotation about the z-axis in order to align the *ENU* frame with the orientation of the *Map* frame.

$$\boldsymbol{T}_{M}^{U} = \begin{bmatrix} \cos(\alpha^{U}) & -\sin(\alpha^{U}) & 0 & 0 \\ \sin(\alpha^{U}) & \cos(\alpha^{U}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.11}$$

In the end we have obtained the second transformation $\boldsymbol{T}_{M}^{E}$ as a chain multiplication of the presented matrices.

$$\boldsymbol{T}_{M}^{E} = \boldsymbol{T}_{A}^{E} \cdot \boldsymbol{T}_{\phi}^{A} \cdot \boldsymbol{T}_{\lambda}^{\phi} \cdot \boldsymbol{T}_{U}^{\lambda} \cdot \boldsymbol{T}_{M}^{U} \tag{5.12}$$

Knowing this homogeneous transformation matrix we can express a generic point in Map $\boldsymbol{p}^M$ into ECEF coordinates.

$$\boldsymbol{p}^E = \boldsymbol{T}_M^E \cdot \boldsymbol{p}^M \tag{5.13}$$

5.3 Pseudorange Constraints in Wolf

The class `ConstraintGPSPseudorange2D` implements in the method `operator()` the math needed to add a constraint to the optimization problem.

Each parameter (or unknown) to be estimated by the optimization problem is called a *State Block*, and it is contained in a Wolf class called `StateBlock`. This class is composed by a vector of variables that have to be estimated. A state block contains also the flag `fixed` that allows to enable or disable on-line the parameter in the optimization problem. The solver considers only non-fixed state blocks, and if the flag is activated the values in the state block are considered like constants by the solver.

In table 5.1 all the state blocks involved in GPS's `operator()` are listed.

Quantity	**State block name**	**Dimension**	**Reference**
$\boldsymbol{b}^M$	vehicle_p	2D	Frame Map
θ^M	vehicle_o	1D	Frame Map
$\boldsymbol{m}^E$	map_p	3D	Frame ECEF
α^U	map_o	1D	Frame ENU
$\boldsymbol{S}^B$	sensor_p	3D	Frame Base
t_b	bias	1D	GPS time

Table 5.1: List of state blocks involved in GPS constrains.

Applying the transformations matrices 5.3 and 5.12 seen in section 5.2, we can compute the sensor position with respect to the *ECEF* frame.

$$\boldsymbol{S}^E = \boldsymbol{T}_M^E \cdot \boldsymbol{T}_B^M \cdot \boldsymbol{S}^B = \boldsymbol{T}_B^E \cdot \boldsymbol{S}^B \tag{5.14}$$

Then we can use this quantity to compute the expected pseudorange:

$$\hat{\rho}_j = \|\boldsymbol{S}^E - \boldsymbol{SV}_j^E\| + c \cdot t_b \tag{5.15}$$

where $\boldsymbol{SV}_j^E$ refers to the position of the j-th satellite with respect to the *ECEF* frame.

Afterwards we compute the residual as the difference between the observed pseudorange and the expected one, dividing everything by the standard deviation of the measurement.

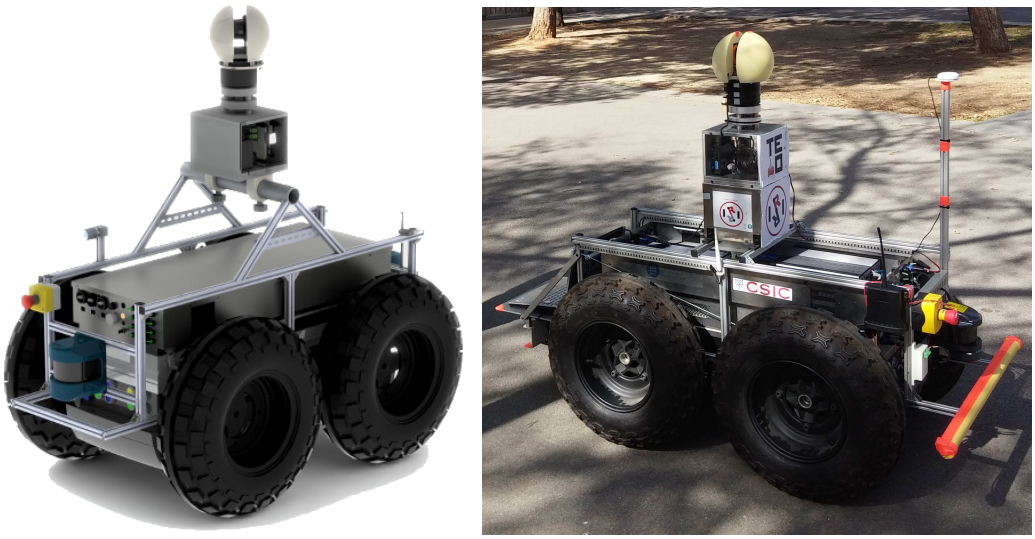
$$e_j = \frac{\hat{\rho}_j - \rho_j}{\sigma} = \frac{\|\boldsymbol{S}^E - \boldsymbol{SV}_j^E\| + c \cdot t_b - \rho_j}{\sigma} \tag{5.16}$$

Note that this residual involves all the quantities that we want to optimize, listed in table 5.1, because $\boldsymbol{S}^E = f(\boldsymbol{b}^M, \theta^M, \boldsymbol{m}^E, \alpha^U, \boldsymbol{S}^B)$.

For each pseudorange measurement available, a constraint is added to the overall optimization problem, possibly with other constraints created by different sensors. All the constraints refer to a state of the robot in a precise time. The solver tries to find the state that minimize the overall mean squared error of the constraints connected to key-frames of the actual window [1].

5.4 Teo Robot and Wheel Odometry

In order to get some real measurements we used an IRI's mobile robot named Teo. Teo Robot [25] is a four-wheeled Unmanned Ground Vehicle



(a) (b)

Figure 5.5: Teo Robot. In (a) we can see its RVIZ model, in (b) there is a photo taken during the experiments, where it is possible to see the mounting point of the external GPS antenna.

[1] The concepts of key-frame and window have been introduced in section 2.3

(UGV) based on the mobile robotic platform Segway RMP 400. The robot is equipped with the electrical system, two Ubuntu-based computers, a battery, a wireless interface etc. At perception level, Teo is equipped with a set of sensors that allow it to observe the environment. The main ones are a front and a rear laser scanners, a pair of stereo cameras, a 3D laser, a GPS receiver, a shaft encoders and an Inertial Measurement Unit (IMU).

A shaft encoder is not an "absolute" encoder, but it is just a "relative" encoder. So the raw measurements are counts of pulses caused by angular increments of the shaft, which jointly with vehicle kinematics allow to estimate the 2D twist (v_x, v_y, ω). Integrating the twist over time allows us to estimate a position of the robot. Using the data produced by these devices it is possible to estimate changes in position over time.

Wheel odometry is the process of integrating the 2D twist produced by shaft encoders, which results in a track of vehicle 2D poses referenced to an initial origin frame, usually called odometry frame.

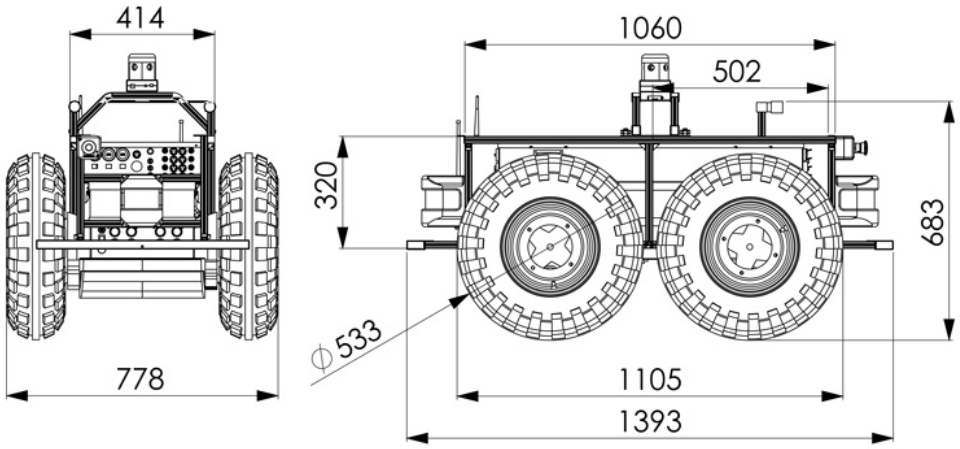


Figure 5.6: Scheme of Teo Robot with dimensions.

Teo is ROS-based robot and there is a ROS package [26] that contains all the nodes needed to handle it, in terms of hardware acquisition and driving.

It is manually teleoperated by a human with a Wii controller and the structure with big wheels (Figure 5.5) permits to use it also in rough grounds.

The Segway platform provides wheel odometry measurements, improved by using an IMU. The IMU sensor uses accelerometers to accurately detect the current rate of acceleration, and gyroscopes to detect changes in rotational attributes like pitch, roll and yaw.

In Teo's two-dimensional current odometry implementation, the straight movement Δx is estimated by wheel odometry, and the yaw $\Delta\theta$ from the IMU's gyros.

All the data received by the sensors is published on different ROS topics, and the two computers mounted on board allows to process it on-line or record it for off-line processing.

5.5 Fusion Between Raw GPS and Odometry

Wheel odometry offers a good estimation of movements in short time, but pose error irremediably drifts with time, due to integration of noisy data with time. On the other hand, GPS accuracy is poorer, but it is bounded over time. The fusion of these two sensors overcomes the performance issues found in each individual sensor, and produces a better localization than the one offered individually during the time.

The GPS can bound the error produced by odometry in the long term, and also positions the trajectory in an absolute way with respect to the Earth, and not only with respect to the Map frame.

Odometry instead overcomes GPS weaknesses like integrity of the solution or interruption of the satellite signal due to obstructed line of sight or multipath reflections.

Moreover, with the approach of not using GPS fix but raw pseudorange data, the system is robust also in case of shading of some satellites. In fact, also if we don't receive enough pseudoranges for computing a GPS fix, they can still constrain the optimization problem, at least partially.

For these reasons, and also to be ready to fuse with other sensor modalities such as lasers or cameras, we decided to use Wolf to fuse the raw GPS pseudorange measurements with Teo's odometry. We created a ROS node, `wolf_ros_gps`, that uses the Wolf library to fuse these two sensing modalities.

The constructor of `wolf_ros_gps` is appointed to create and initialize the Wolf tree. It has to create the root node `WolfProblem` and add in the hardware branch the two sensor models we want to use, `SensorGPS` and `SensorOdom`, and the corresponding processors.

To help the solver converging, it is better to initialize the Map position with respect to ECEF with a value close to the real one. This can be done using the GPS fix, if available, otherwise it is possible to discover it using a map, such as Google Maps.

The constructor initializes also Ceres-Solver through the `CeresManager` class. This object is responsible to keep the Ceres optimization problem up-

dated, with all the fixed and non-fixed state blocks inside the actual window.

When GPS or odometry measurements are published by the sensors, the respective callbacks deal with the creation of the capture objects and the addition of them in the current Wolf frame.

Odometry is published by Teo in a specific topic in the form of transformations between the parent frame *Odom* and the frame *Base*.

When we create a new frame, to help the convergence of the solution we want to have an initial pose guess as close as possible to the real pose of the robot in that moment. In order to do that, if there is a previous key-frame, we select its pose as a guess for the new one. If it also has an odometry capture, we integrate it through time until the current timestamp and we have a better pose guess. Every time a new frame is added to the trajectory we have to move the window and, if the window is not big enough, we have to remove the oldest frame. This frame is only removed from the window, and not from the wolf problem. This means that its position and orientation will be considered estimated, and the constraints associated to it will not affect the following optimizations.

Every time we want to process the arrived data we have to update the optimization problem and then solve it. Both operations are done with `CeresManager`.

The obtained solution after the optimization is the robot pose with respect to Map frame, $\boldsymbol{T}_B^M$. This pose is represented by frame *Base*. This frame can only have a parent, that is the *Odom* frame, because it is published by Teo. Because of that, we cannot publish directly the pose of *Base* with respect to *Map*. For this reason we have to make *Odom* child of *Map*, and publish a transform between this two frames in order to have Base in the same pose. In Figure 5.7 we can see the relation between these frames. The pose of *Base* published by the odometry is $\boldsymbol{T}_B^O$.

Through Ceres-Solver we calculate the pose of Base with respect to the map, $\boldsymbol{T}_B^M$. To move frame *Odom* from *Map*, in order to have $\boldsymbol{S}^O = \boldsymbol{S}^M$, we need to calculate

$$\boldsymbol{T}_O^M = \boldsymbol{T}_B^M \cdot \boldsymbol{T}_O^B = \boldsymbol{T}_B^M \cdot (\boldsymbol{T}_B^O)^{-1}$$

At the beginning of the experiment the frame *Map* will perfectly overlap the *Odom* frame, and during the time they may separate. The space between this two frames represents the drift of the odometry positioning respect to the positioning estimated with sensor fusion.

To connect these frames to *ECEF* frame we need also to publish a transform between *ECEF* and *Map* frames. To do that we use the *Map* position and orientation estimated and, with the math we have seen in equations 5.3 and 5.12, we can publish the transformation $\boldsymbol{T}_M^E$

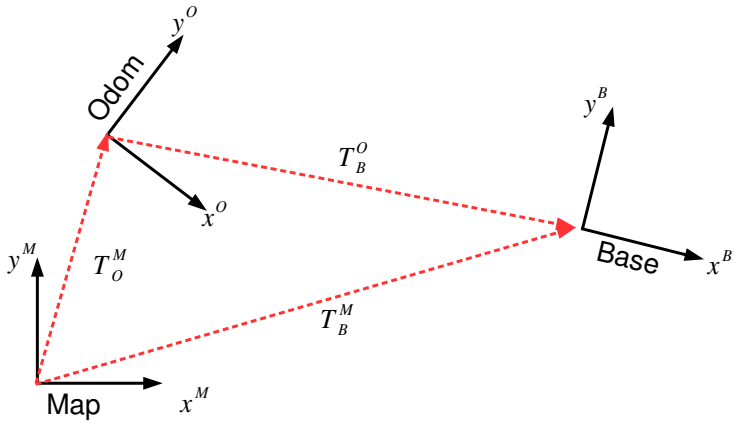


Figure 5.7: Relations between *Map*, *Odom* and *Base*.

RVIZ will listen to the transformations published for every key-frame and visualize it, as we will see in the section 5.6,

In Figure 5.8 we can see the Wolf's tree specialized in the implementation of raw GPS pseudoranges and odometry.

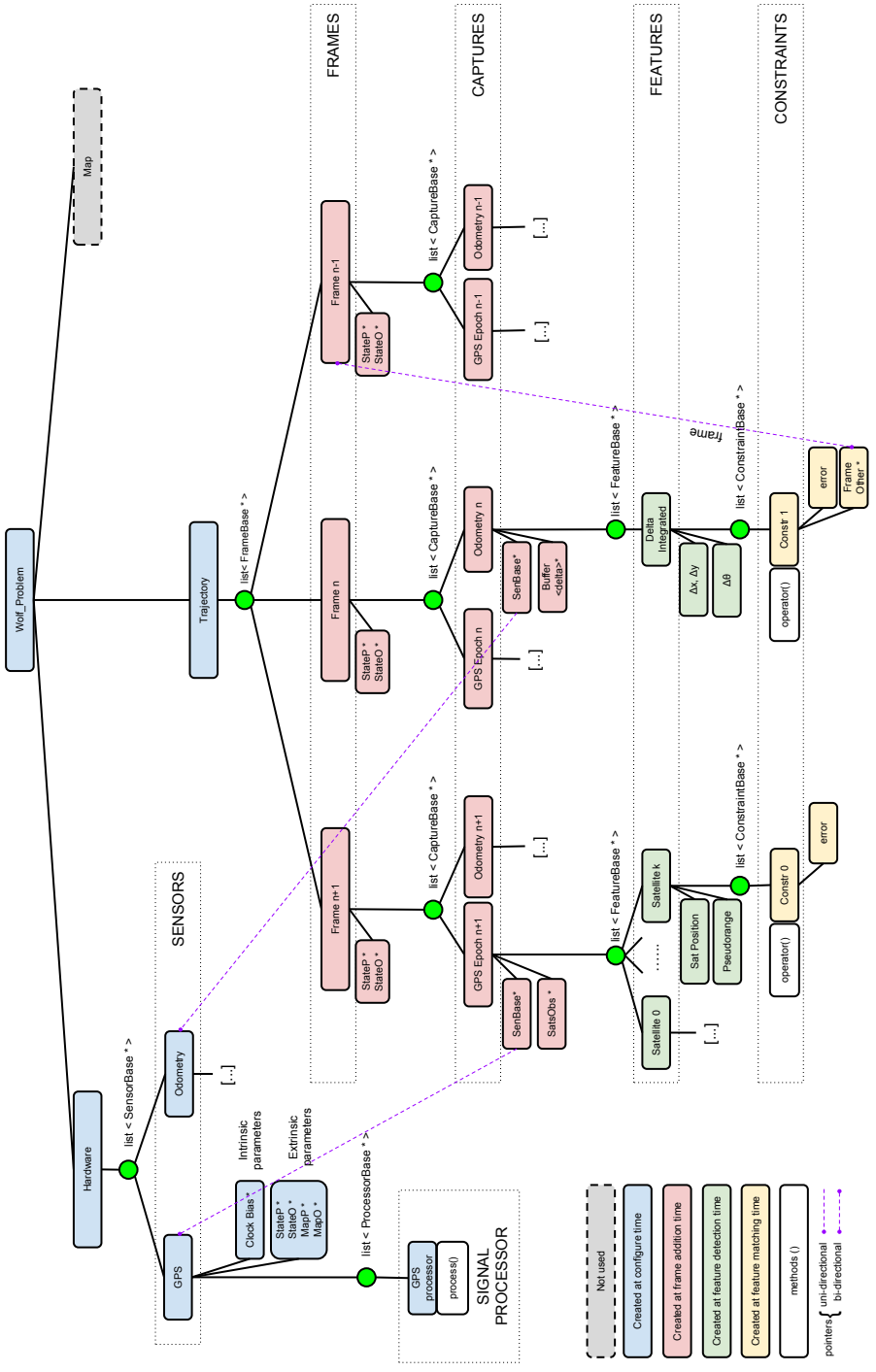
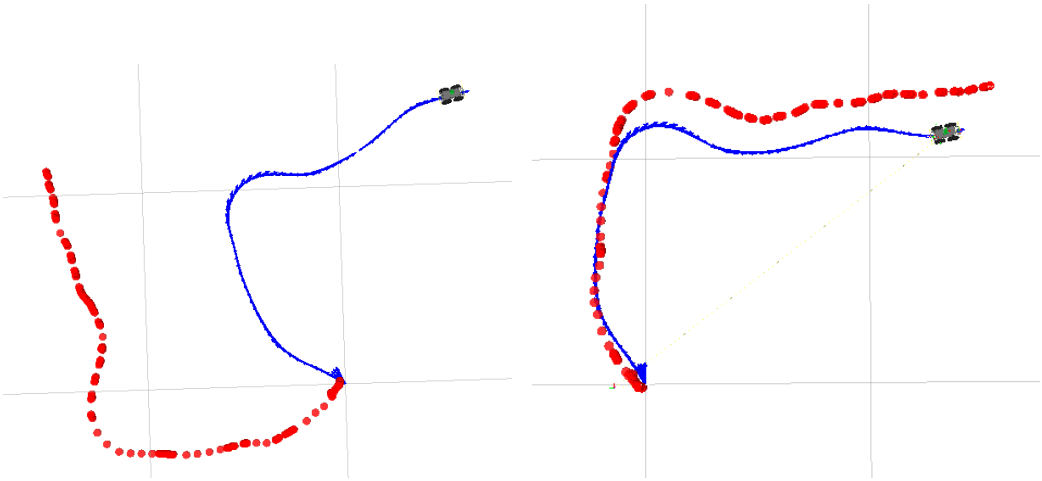


Figure 5.8: WOLF tree specialized with raw GPS and odometry. We are not mapping, so the *Map* branch is dashed.

5.6 Sensor Fusion Results

In the experiments we made, we assumed to be able to estimate the ECEF coordinates where the experiment starts, in order to initialize correctly the map position $\boldsymbol{m}^E$. This assumption is not too much restrictive, because in outdoor vehicle localization we can obtain a GPS fix, and if it is not available it is possible to have a guess of the initial position using other instruments, such as Google Maps.

If the initial position is easy to obtain precisely, the map orientation α^U is definitely not. Using an incorrect value for this state block leads to a bad estimation of the vehicle position with respect to the ECEF coordinates. With the sensor fusion we did through Wolf, the map orientation α^U can be estimated correctly, and even if it has been initialized randomly or with a value really far from the real one, it immediately converge to the correct orientation, as we can see in Figure 5.9.

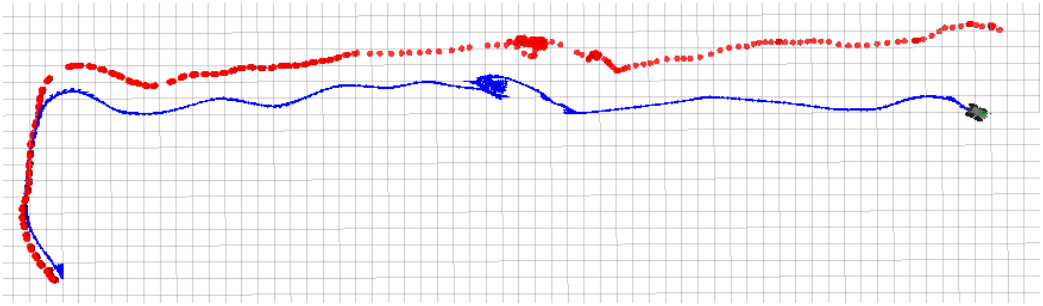


(a) (b)

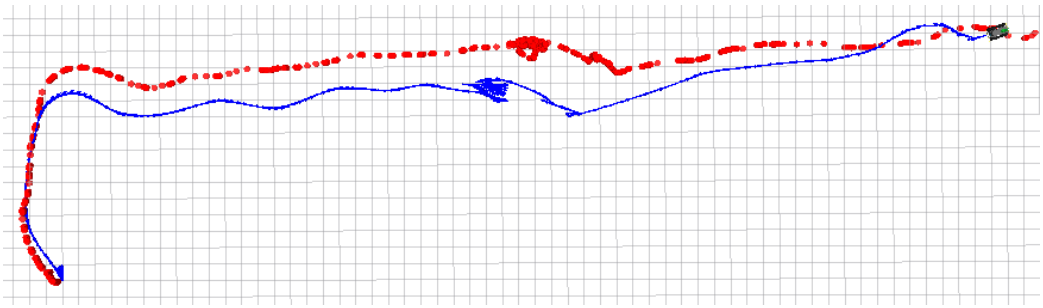
Figure 5.9: Correction of α^U. These figures plot the trajectory of the robot, in blue, and the GPS fixes produced by the GPS receiver mounted on the robot, in red. (a) shows a comparison between the two positioning without using an optimized value of α^U, (b) shows the positioning obtained using sensor fusion.

The first goal we achieved is the absolute positioning of the map frame and the trajectory with respect to the Earth.

Figure 5.10(a) shows the whole trajectory, computed using only odometry, but fixing the map orientation α^U with the correct angle we estimated before.



(a) Robot trajectory obtained using only odometry measurements.



(b) Robot trajectory obtained fusing odometry and raw GPS measurements.

Figure 5.10: In blue is represented the robot's trajectory estimated by Wolf. In red there are represented the GPS fixes as comparison. The trajectory is proportional to the grid, whose cells are squares with sides of 1 meter.

At the end of the path the estimated position of the robot is more that 5 meters far from the GPS fix produced by AsteRx1. We can also notice how the odometry estimates shorter distances respect to the GPS fixes.

In Figure 5.10(b) we can see the trajectory computed by Wolf fusing odometry with raw GPS. Here the GPS constraints remove the drift of the odometry, and the final part of the trajectory is approximately 2.5 meters far from the GPS fix.

We have noticed that the distance covered according to the odometry sensor is in general shorter than the distance we obtain relying on GPS fix. In our experiment the trajectory is dominated by the odometry, but gradually merges to the measurements produced by GPS. Fusing these kind of discording data possibly leads to problems in the estimation of the fused trajectory. A more appropriate calibration of robot odometry system would also be necessary to improve results.

The restriction on a plane we made in section 5.2 is fundamental to make cooperate a sensor like the GPS, that produce a three-dimensional position-

ing, with a sensor like wheel odometry that lie in a bi-dimensional space.

Anyway, the assumption of working on a plane tangent to the WGS84 geoid has proved to be too restrictive and creates errors in the estimation of the trajectory. This is accentuated because in the surrounding of the laboratory, where we did the experiment, there is a slope. As we can see from Figure 5.11 the GPS fixes start in the correct position [2] but soon they assume negative values in z^M and they go under the Map plane.

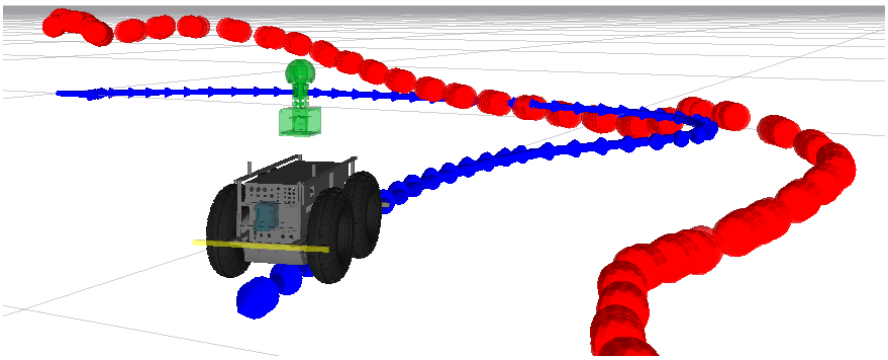


Figure 5.11: Focus on the altitude problem, caused by the plane of the experiment differently inclined with respect to Map plane.

The sensor fusion we did is robust against interruption of GPS data, due to sensor breakage or obstructed line of sight between satellites and GPS antenna.

We artificially limited the number of satellites available during our experiments, to see how it deals in situations where reception is not perfect for the GPS antenna, but still can receive some GPS data. In these cases, if less that 4 satellites were available, a computation of the GPS fix would be impossible. With our sensor fusion this problem overcame, because we can rely on the wheel odometry. We did a lot of tests, from which emerged that even from only two satellites measurements we still can notice a slow convergence of the trajectory to the path indicated by the GPS fixes. With only one measurements, in our experiments, we didn't noticed any effect, and the trajectory is completely dominated by odometry measurements.

[2]The GPS antenna is mounted in Teo 110 centimeters high from the ground.

6. Conclusions

In this thesis we learned the importance of simultaneous localization and mapping in mobile robotics. We focused on the localization part and we deepen on the GPS, learning how geometrical constraints created by GPS pseudorange measurements can be added to a SLAM optimization problem.

With this Master's Thesis work we achieved the following results:

- We developed a software framework that processes raw GPS data produced by a real receiver and creates geometrical constraints for a localization problem.
- We integrated our work in Wolf, a software framework for managing SLAM, enriching its sensor fusion capabilities with the possibility of including geometric constraints related to GPS pseudorange measurements.
- With this software environment we have been able to test the fusion between data coming from wheel odometry and raw GPS pseudorange measurements. This sensor fusion produced some positive results:
  - It has been possible to position the robot in an absolute way respect to ECEF coordinates, and to estimate a trajectory that depends on both sensors.
  - The GPS allows to bound the error of the wheel odometry, that otherwise will irremediably drift during time.
  - On the other hand, odometry allows to continue the estimation of the vehicle's trajectory in zones where the GPS reception is poorly or not available.
  - The trajectory produced with this sensor fusion is smoother compared to the one produced by trilaterating raw GPS pseudoranges.

6.1 Future developments

The solution we proposed can still be improved. We identify some points in which we should work in future developments, in order to further improve this solution:

- We observed during the fusion experiments that our solution is still sensitive to outliers. Some wrong estimations of satellite positions or the associated pseudorange measurements can cause a wrong pose in Wolf's trajectory that the odometry is not able to overcome yet. We must enhance outlier detection, to ensure that the solver discards the faulty measurements.

- If a sensor is producing a high number of outliers it is also possible to assume that this particular sensor is in a faulty situation (i.e. it is obstructed or there are high interferences), and rely more on the measurements produced by the other sensors. This should be done in the Wolf implementation and in the solver settings.

- Diminish the number of outliers, improving satellites positioning and pseudoranges correction:

  - We can intervene at raw GPS data processor nodes level, applying advanced techniques used in commercial devices for decrease errors in GPS measurements, such as SBAS augmentation cited in section 4.5 and multi-path correction.
  - We should use a dual-frequency GPS receiver. Using two frequencies we could eliminate (to the first order) the ionospheric delay through a linear combination of the L1 and L2 observations (see equation 3.10).

- The environment where we want to localize our vehicles must be taken in account too, because the proposed solution assumes to work in a plane. If the the environment where the robot is moving is steep, or not flat, it must be considered in the formulation of the map. In this latter case we should fuse with 3D odometry.

- Use informations about the user velocity: we could estimate the velocity from the raw data produced by the GPS receiver, using Doppler measurements related to user-satellite motion. Doppler frequency shifts of the received signal, produced by user-satellite relative motion, enables velocity accuracy of a few centimeters per second. When we started

our work, Wolf did not take into account the velocity state of the vehicle, but now this functionality has been implemented and we could use the above-cited method to take advantage also of these Doppler measurements.

Bibliography

[1] "CARGO-Ants web page." http://www.cargo-ants.eu/.

[2] M.W.M.G.Dissanayake, P.Newman, S.Clark, H.F.Durrant-Whyte and M.Corsba, "A Solution to the Simultaneous Localisation and Map Building (SLAM) Problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms.," *Robotics and Automation Magazine*, June 2006.

[4] G.Grisetti, R.Kummerle, C.Stachniss, W.Burgard, "A tutorial on graph-based SLAM," *Magazine on Intelligent Transportation Systems*, 2010.

[5] J. Solà, "Course on SLAM." http://www.iri.upc.edu/people/jsola/JoanSola/objectes/toolbox/courseSLAM.pdf, Jul 2015.

[6] R. Kümmerle, B. Steder, C.Dornhege, M.Ruhnke, G.Grisetti, C.Stachniss, and A.Kleiner, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, vol. 27, pp. 387–407, 2009.

[7] "WOLF: a versatile framework for localization and mapping." https://github.com/IRI-MobileRobotics/wolf.

[8] S. Agarwal, K. Mierle, and Others, "Ceres Solver." http://ceres-solver.org.

[9] K. E. Foote and D. J. Huebner, "Error, Accuracy, and Precision," tech. rep., Department of Geography, University of Texas at Austin, 1995.

[10] Hoffman-Wellenhof, B., H. Lichtenegger and J. Collins., *Global positioning system: theory and practice.* 2001.

[11] Elliott D. Kaplan, Christopher Hegarty, *Understanding GPS: Principles and Applications, Second Edition.* 2005.

[12] European Space Agency - NAVIPEDIA, "GPS Signal Plan." `http://www.navipedia.net/index.php/GPS_Signal_Plan`.

[13] European Space Agency - NAVIPEDIA, "GPS Navigation Message." `http://www.navipedia.net/index.php/GPS_Navigation_Message`.

[14] D.J. Allain and C.N. Mitchell, "Ionospheric Delay Corrections for Single-Frequency GPS Receivers over Europe Using Tomographic Mapping," *GPS Solutions*, vol. 13, pp. 141–151, 2009.

[15] Open Source Robotics Foundation., "ROS.org web page." http://www.ros.org/.

[16] B. Tolman, R. B. Harris, T. Gaussiran, D. Munton, J. Little, R. Mach, S. Nelsen, and B. Renfro, "The GPS Toolkit: Open Source GPS Software," in *Proceedings of the 16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, (Long Beach, California), September 2004.

[17] Global Positioning Systems Directorate Systems Engineering & Integration, "Navstar GPS Space Segment/Navigation User Interfaces(IS-GPS-200H)." http://www.gps.gov/technical/icwg/IS-GPS-200H.pdf, September 2013.

[18] Gurtner, W., "RINEX, The Receiver Independent Exchange Format." `https://igscb.jpl.nasa.gov/igscb/data/format/rinex300.pdf`, November 2007.

[19] Septentrio satellite navigation, *Command Line Interface Reference Guide*, 2008.

[20] Septentrio satellite navigation, *SBF Reference Guide*, 2008.

[21] Septentrio satellite navigation, *AsteRx Firmware User Manual*, 2007.

[22] "SBAS Fundamentals." `http://www.navipedia.net/index.php/SBAS_Fundamentals`.

[23] Joe Mehaffey, "GPS Altitude Readout: How Accurate?." `http://gpsinformation.net/main/altitude.htm`.

[24] Corominas Murtra, A., *Map-based Localization for Urban Service Mobile Robotics*. PhD thesis, Universitat Politècnica de Catalunya, 2011.

[25] IRI Robotics Lab, "Teo Robot." `https://wiki.iri.upc.edu/index.php/TEO`.

[26] IRI Robotics Lab, Marti Morta, "ROS package teo_robot." `http://wiki.ros.org/teo_robot`.

Appendices

Source Code Repositories

All the software nominated in this thesis is open-source and hosted in various repositories online.

- Software fully developed by me for this Master's Thesis:

Name	Description	Repository
wolf_ros_gps	ROS wrapper of wolf with GPS and wheel odometry	`https://github.com/IRI-MobileRobotics/wolf_ros_gps`
trilateration	Trilateration library	`https://github.com/pt07/trilateration`
raw_gps_ros	GPS tools ROS	`https://github.com/pt07/raw_gps_ros`
raw_gps_utils	External library with data structures used by Wolf for GPS data	`https://github.com/pt07/raw_gps_utils`
rinex_reader	Library that parse rinex files and manage the data	`https://github.com/pt07/rinex_reader`
trilat_node	ROS wrapper of rinex reader library	`https://github.com/pt07/trilat_node`

- Repositories where I contributed:

Name	Description	Repository
wolf	Wolf library	`https://github.com/IRI-MobileRobotics/wolf`
iri_asterx1_gps	ROS wrapper of the low level driver for Septentrio AsteRx1 GPS receiver	`https://devel.iri.upc.edu/labrobotica/ros/iri-ros-pkg_hydro/metapackages/iri_common_drivers/iri_asterx1_gps/`

iri_common_drivers_msgs	ROS messages used for GPS data	`https://devel.iri.upc.edu/labrobotica/ros/iri-ros-pkg_hydro/metapackages/iri_common_drivers/iri_common_drivers_msgs/`
teo_robot	ROS nodes for control Teo Robot	`https://devel.iri.upc.edu/labrobotica/ros/iri-ros-pkg_hydro/metapackages/teo_robot/`