Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'Informazione

A General Sensor-fusion and Parameters Self-calibration Framework with Applications in Mobile Robotics

Tesi di dottorato di:
Davide A. Cucci

Relatore:
Prof. Matteo Matteucci
Tutore:
Prof. Andrea Bonarini
Coordinatore del programma di dottorato:
Prof. Carlo Fiorini

2014 — XXVII ciclo

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci, 32 I 20133 — Milano

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'Informazione

A General Sensor-fusion and Parameters Self-calibration Framework with Applications in Mobile Robotics

Doctoral Dissertation of:
Davide A. Cucci

Advisor:
Prof. Matteo Matteucci
Tutor:
Prof. Andrea Bonarini
Supervisor of the Doctoral Program:
Prof. Carlo Fiorini

2014 — XXVII edition

The justification for a university is that it preserves the connection between knowledge and the zest of life, by uniting the young and the old in the imaginative consideration of learning. Youth is imaginative, and if the imagination be strengthened by discipline this energy of imagination can in great measure be preserved through life. The tragedy of the world is that those who are imaginative have but slight experience, and those who are experienced have feeble imaginations. Fools act on imagination without knowledge; pedants act on knowledge without imagination. The task of a university is to weld together imagination and experience.

A.N. Whitehead, 1967

Acknowledgements

This work has been partially supported by research grants within the following research project:

> "ROAMFREE: Robust Odometry Applying Multi-sensor Fusion to Re- duce Estimation Errors" PRIN 2009 grant by the Italian Ministry of University and Research (MIUR)

Abstract

Autonomous robots use sensors to acquire knowledge about the state of the world, in particular to reduce their uncertainty about critical variables related to the assigned tasks. When multiple sensors observe different aspects of the reality, they report noisy, overlapping, possibly contradictory measurements that have to be properly processed to update the robot internal belief. In this work we introduce ROAMFREE, a general, open-source, framework for multi-sensor fusion and parameter self-calibration in mobile robotics. A comprehensive logical sensors library allows to abstract from the actual hardware and processing while preserving modeling accuracy thanks to a rich set of calibration parameters (e.g., sensor geometric placement, biases, gains and distortion matrices). A modular formulation of the information fusion problem has been obtained based on state-of-the-art factor-graph inference techniques; it allows to handle arbitrary number of multi-rate sensors and to adapt to virtually any kind of mobile robot platforms, such as Ackerman steering vehicles, quadrotor unmanned aerial vehicles, omni-directional mobile robots. Different solvers are available to target both high-rate online pose tracking tasks and offline accurate trajectory smoothing and parameter self-calibration. An extensive evaluation of the resulting framework has been performed on different mobile robots. ROAMFREE has already proved its flexibility and out-of-the-box deployment in several, real-world, information fusion and sensor self-calibration problems.

Contents

Chapter 1

Introduction

Robots are mechanical or virtual artificial agents able to perform given tasks with a certain degree of autonomy. These tasks always involve interaction with the environment, whatever it is our familiar physical world or some virtual scenario.

In principle, most of these tasks could be performed relatively easily if only the robot knew *certain* quantities such as its own position, the position of its goal, the current distance from walls, if the planned path towards the goal is free from obstacles, and where possible obstacles are located. Unfortunately, in practice these variables are seldom directly observable. Moreover, even in scenarios where the operating conditions, such as the light conditions, or the site map, can be controlled or *jointly* designed with the robotic system, there will always be inescapable degrees of uncertainty in the robot and environment state.

In order to bound the uncertainty in their knowledge, most of the modern robots employ *sensors* and maintain an internal model of the state of the world; this model is updated according to observation evidence and it is then employed to make decisions about how to accomplish the assigned tasks. The set of sensors available for the robot is always chosen depending on the operating environment and on the required degree of autonomy. In general, as the assigned tasks grow in complexity, the set of variables that have to be observed increases and multiple sensors are required. As heterogeneous sensors observe different aspects of the

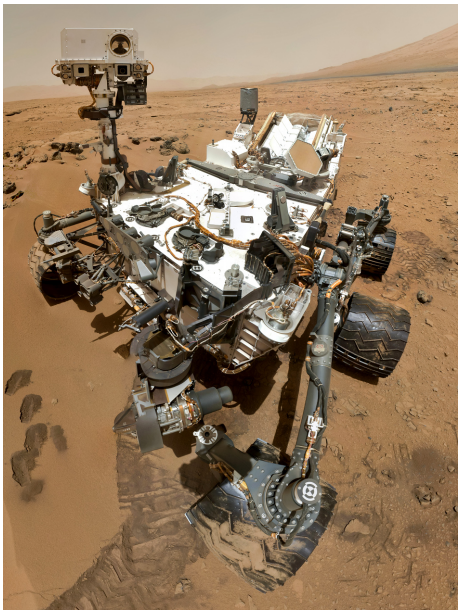


Figure 1.1.: A *selfie* of the Curiosity Rover.

reality, redundancy in perception results in an increased fault tolerance and robustness with respect to unforeseen situations.

As an example, let us consider the case of robotic systems for space exploration, such as the Curiosity rover depicted in Figure 1.1, and part of the Mars Science Laboratory mission. Because of the delay in communication between Earth and Mars, the robot can not be teleoperated, and waiting for human instructions at each unforeseen situation is clearly impractical. Thus, the robot needs a high degree of autonomy, at least for elementary tasks such as heading towards given positions and obstacle avoidance. To this end, several sensors are employed; leaving apart the scientific experiments and general purpose elements such as the *mastcam*, a multi-spectra, high definition camera, the rover has two pairs of *navcams*, to acquire stereoscopic 3-D images, plus four pairs of *hazcams*, which are used for autonomous hazard avoidance and safe positioning of the robot arm, for a total of twelve cameras employed in navigation, plus an inertial measurement unit.

As new, noisy, possibly contradictory, evidence comes from multiple, heterogeneous, sensors, processing has to be applied in order to *fuse* the available information and update the robot internal model of the world. This is model is often called *belief*, and in modern robots it also includes an explicit characterization of the uncertainty regarding critical variables. The problem of how these internal believes can be consistently

updated as new observations become available has been subject of active research in the last fifty years, and it is still ongoing. Many techniques have been proposed and effectively employed in several applications. Notable examples are the Extended Kalman Filters, or, in general, recursive Bayes filters, and, more recently, graph-based optimization techniques.

However, hardware sensors, or pre-processing to be applied on raw data, often involve calibration or tuning parameters that turn out to be critical to build internal robot believes. For instance, on a mobile robot with a single camera, it is impossible to estimate the robot velocity from successive frames unless we know the orientation of the camera with respect to the robot base. Other examples are gains and biases in inertial measurement units, ferromagnetic properties of the robot affecting magnetometer readings, intrinsic matrices and depth distortion pattern of a RGB-D cameras, to name a few. To ensure that sound and consistent state estimation can be achieved, it is often required to determine these parameters with a high degree of accuracy. However, it is often difficult to to determine these by directly inspecting the robot (think about the case of 3-DoF orientations), while others are simply not directly observable, e.g., the matrices of intrinsic camera parameters.

A number of ad-hoc solutions has been proposed in the literature to handle accurate calibration of very specific sensor configurations (and they are still subject of active investigation). These techniques often rely on artificial environment structures, such as checkerboards in camera calibration, or on the availability of external information, not produced by the set of sensors being calibrated, such as position ground truth. Unfortunately, relevant parameters might change over time, such as biases in gyroscope sensors, which depend on environment temperature, motion, and on a number of other factors. In this cases, offline, ad-hoc, calibration procedures, and environment structures, can not be employed and typical solutions require the robot state space to be augmented to include estimates for calibration parameters. Few work has been done on the *self-calibration* of an arbitrary set of sensor, i.e., the problem of determining sensor calibration parameters employing only the information produced by the sensors themselves, possibly, during normal robot operation.

As robotic systems face new and more advanced tasks, system developers and researchers are required to handle very complex sensor-fusion and parameter calibration problems. Despite the wide variety of solutions available in the literature, platform dependent specifications make them not directly applicable, or require adaptations, enhancement, or substantial extensions. The lack of off-the-shelf, flexible solutions which are deployable with minor effort undermine the availability of base-

line solutions to compare new approaches against and often requires researchers to develop from scratch even very simple sensor-fusion algorithms, "reinventing the wheel" and scarifying reusability.

In this work we propose a general and flexible approach to the problem of multi-sensor fusion and parameters self-calibration, resulting in a software framework which can be deployed on very different robotic platforms and sensor configurations. In its development, we employ and extend mathematical and software engineering techniques to ensure that the resulting framework can be easily specialized to handle specific cases, and some of its component replaced without any change to the overall system architecture. We believe that our approach could significantly reduce the effort needed for developing new robotic applications, boosting research and easing the comparison of different approaches.

1.1. Main Contributions

In the following we summarize the main contributions of this work and we provide references to international peer-reviewed publications in which we discuss our specific contributions in details.

A modern sensor-fusion approach. A state of-the-art formulation of the problem of simultaneous multiple sensor information fusion and parameter self-calibration is presented, which is based on factor-graphs and non-linear, manifold-aware, max-likelihood estimation. Decoupling is achieved between state representation, sensor models, and solver algorithms, so that each of these components can be extended or replaced requiring no change in the rest of the architecture.

Davide Antonio Cucci and Matteo Matteucci. On the development of a generic multi-sensor fusion framework for robust odometry estimation. *Journal of Software Engineering for Robotics*, 5(1):48–62, 2014

A library of reusable sensor models. These models characterize observations with respect to their measurement domain, e.g., position, velocity, orientation, etc., rather than directly describing hardware sensors. Moreover, each model comes with a set of calibration parameters to accurately characterize a wide variety of information sources. A decoupling layer is also developed to make sensor models independent from the specific representation of the robot state.

Davide Antonio Cucci and Matteo Matteucci. Position tracking and sensors self-calibration in autonomous mobile robots by gauss-newton

optimization. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1269–1275. IEEE, 2014

Davide Antonio Cucci and Matteo Matteucci. A flexible framework for mobile robot pose estimation and multi-sensor self-calibration. In *Informatics in Control, Automation and Robotics (ICINCO), 2013 International Conference on*, pages 361–368, 2013

An outlier rejection mechanism. It is based on a RANSAC approach and allows to exclude spurious observations before they are employed in estimation, increasing robustness beside accuracy. A low-dimensional kinematic model is employed to approximate the robot trajectory in a local time window. Multiple hypotheses are generated based on random, minimal, set of observations and they are ranked according to the other available readings.

An *off-the-shelf* software framework. It provides a simple yet powerfull C++/Python API and seamless integration with the popular Robot Operating System (ROS) framework and other open-source and open-hardware tools such as the Rapid Robot Prototyping toolkit.

Davide Antonio Cucci, Martino Migliavacca, Andrea Bonarini, and Matteo Matteucci. Development of mobile robots using off-the-shelf open-source hardware and software components for motion and pose tracking. In *Intelligent Autonomous Systems (IAS), 2014 International Conference on*, page to appear, 2014

A set of physical testbeds for perfomance evaluation. The proposed framework is currently employed as the pose-tracking and calibration component on two autonomous mobile robots: the Quadrivio all-terrain ATV and the Lurch Autonomous Wheelchair. Moreover, odometry and pose tracking benchmarks have been carried on other two modile robots: the heavy duty differential drive Robocom and Triskar2, an omnidirectional wheeled robot designed for aggressive maneuvers in indoor environments.

Gianluca Bardaro, Davide Antonio Cucci, Luca Bascetta, and Matteo Matteucci. A simulation based architecture for the development of an autonomous all terrain vehicle. In *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 74–85. Springer International Publishing, 2014

1.2. Thesis Outline

This work is divided in two parts and the aforesaid contributions are presented according to the following structure.

Chapter 1 has briefly introduced the information fusion and parameters self-calibration problem in mobile robotics, the motivations and the main contributions of this work.

PART I - Framework Description

Chapter 2 provides a discussion about the state of the art in state estimation in mobile robotics from an historical perspective, highlighting the main ideas and current open issues.

Chapter 3 introduces the ROAMFREE framework architecture, presenting its main components and the techniques employed to achieve the decoupling between state representation, error models and solvers from a system level perspective.

Chapter 4 details the mathematical formulation of the state variables operator interfaces, the decoupling layer between state representation and sensor error models, and the sensor models themselves.

Chapter 5 presents the RANSAC inspired outlier rejection algorithms employed to detect spurious sensor readings by means of multiple trajectory hypotheses evaluation.

PART II - Experimental Evaluation

Chapter 6 begins the ROAMFREE experimental evaluation, considering the Quadrivio robot, an all-terrain autonomous vehicle. Here we evaluate the sensor parameter parameter self-calibration capabilities of the proposed framework and we discuss results regarding its usage within a trajectory control loop.

Chapter 7 explores sensors self-calibration considering the Lurch Autonomous Wheelchair. A simplified formulation of the Simultaneous Localization and Mapping problem is addressed employing fiducial markers as landmarks. The geometric placement and odometry calibration parameters of the robot are retrieved along with landmark positions.

Chapter 8 discusses use cases in which the proposed framework is employed in online pose tracking for two different wheeled robots, demonstrating an *off-the-shelf* deployment. Examples of the API usage are discussed along with integration with ROS and R2P, a framework for rapid robot prototyping.

Chapter 9 includes conclusions and a discussion of the issue still open and proposals for future developments.

Part I.

Framework Description

Chapter 2

Background

From the nineties onwards, it was generally understood that maintaining a single hypothesis on the world state is not enough in practice: a characterization of the uncertainty in the robot knowledge is also needed. This idea is well discussed from Thrun et Al., in their famous book *Probabilistic Robotics* [78] and it is summarized by the following conjecture:

> *"A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not."*

To see the rationale behind this statement, let us consider the following example in which a mobile robot has to pass through a door, which can be either open or closed. The robot senses the environment and forms its own *belief* accumulating evidence regarding the status of the door. Suppose that at a certain point, according to past observations, the probability that the door is open is only slightly greater than closed. For instance, people moving in its neighborhood might have tricked the robot range sensors. In this situation a probabilistic robot would approach the door slowly, in the hope that future observation will support the conjecture that the door is open. Conversely, a robot which does not take into account uncertainty in its beliefs would just assume that the door is open and proceed at cruise speed towards the door, eventually resulting in a spectacular crash.

Following this idea, most of the modern autonomous robots maintain *believes* over the state space, i.e., the domain of the critical variables

for which an estimate has to be maintained, in terms of probability distributions so that a probability value is associated to each state. Thus, the robot never rely on a single "best guess" as to what might be the case in the world, yet it exploits some representation of the uncertainty in its knowledge, being able to distinguish between facts that are likely and facts for which little or no knowledge is available.

The problem which arises is how to update this belief distribution as new observations are available. Moreover, multiple sensors are often employed in a redundant fashion, or heterogeneous ones observe different effects related to the same state variable, with the aim of reducing measurement uncertainty and increase the robustness of the system. How can raw sensor data spanning different measurement domains be integrated, or *fused*, in a probabilistic fashion, contributing in the estimation of the robot belief?

In this work we employ sensor-fusion techniques to update believes regarding robot poses and calibration parameters, relying on multiple, heterogeneous, asynchronous information sources. Before presenting our approach, it seems appropriate to discuss other state estimation techniques employed in robotics and similar works.

2.1. Bayesian State Estimation

In probabilistic robotics, a density is maintained over the state space:

$$bel(x_t) = p(x_t | z_{1:t}, u_{1:t}), \tag{2.1}$$

which is a posterior distribution conditioned on all the measurements z_t, and controls u_t up to time t. Here, with the term *controls*, we intend the system inputs, or, in other words, the variables whose value is under direct control of the robot, such as wheel speed setpoints, joint positions, and so on. We are interested in how the posterior belief, $bel(x_t)$, is computed as new controls and a sensor readings are available.

Belief update is often performed in a recursive fashion, integrating each sensor reading as soon as it is available and producing a *posterior* distribution that represents the probability of each state given its *prior*, the sensor readings and the controls. The resulting distribution becomes the prior for the integration of the next sensor reading. This approach goes under the name of recursive state estimation and its most general form is given by the Bayes filter, which we report here for convenience.

In the first step of the algorithm, we *predict* the next state given u_t, for each possible state x_{t-1}. The second step is referred as the

Algorithm 1 The general algorithm for Bayesian filtering.

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

update, and performs the integration of the sensor reading z_t, effectively producing the posterior belief. This is an application of the Bayes rule, expressing the desired probability density $p(x_t|z_t)$ as a function of $p(z_t|x_t)$. Note that recursive filtering algorithms rely on: (i) the Markov assumption, i.e., past and future data are independent if one knows the current state x_t, (ii) the state is complete, which implies that $p(z_t|x_{0:t}, z_{1:t}, u_{1:t}) = p(z_t|x_t)$, i.e., the current state summarizes all the past inputs and measurements. For details see [78, Chapter 2.4].

Here it is important to remark that Bayesian filters offer a straightforward solution to perform probabilistic information fusion between multiple sensors. Indeed, if we could formulate for each sensor a *measurement distribution*, i.e., $p(z_t|x_t)$, or in other words, if we could associate a probability for the current sensor reading as a function of the current state x_t, then we would just need to apply Algorithm 1 for each sensor.

The Bayes filter algorithm can seldom be applied directly due to difficulties in analytically representing arbitrary multivariate posteriors and solving the integrals involved in prediction. Practical implementations of the Bayes filter rely on assumptions on the prior and posterior distributions, or employ Monte Carlo or other non-parametric representations.

The first practical implementation of the Bayesian filter for continuous domains dates back to 1960 and it is due to Rudolph E. Kalman [50]. The impact of that breakthrough is remarkable, the original paper accounts for more than seventeen thousands citations, putting it among the most referenced in the field. The original formulation of Kalman Filters assumes that belief distributions and measurement noise are Gaussian and that system and observation models are linear. Under this assumptions, the Kalman update equation yield the optimal state estimator, in terms of mean squared-error. If the latter assumption does not hold, system and/or measurement models can be linearized, yielding the Extended Kalman Filters (EKFs); these are still widely employed today and are often the first choice in recursive state estimation. However, none of the optimality proofs for KF hold in the non-linear case.

Researchers have focused on improving the Kalman filter approach, "gnawing" one, or both, of its fundamental assumptions. Indeed, while it is often reasonable to assume that the belief distributions are Gaussian, the propagation of Gaussian noise through highly non-lienar system or observation models by means of linearization, as in EKFs, it is often inaccurate. In 1997, the Unscented Kalman Filter (UKF) has been proposed by Simon Julier and Jeffery Uhlmann [48]. It introduces the so called *unscented transform*, that has been shown to model the propagation of a Gaussian noise through an arbitrary non-linear system model more accurately with respect to linearization, without dropping the property that the posterior distribution is still Gaussian.

Various different approaches have been proposed in case a Gaussian belief fails to model the state uncertainty. Indeed, normal distributions are completely characterized by their mean and covariance, and are well suited to represent *one* hypothesis regarding the state of the world. There are situations in which, after the integration of sensor readings into the belief, the posterior distribution becomes multi-modal. In this case one might desire to maintain multiple hypothesis on the robot state. A possible solution is given by Multi Hypothesis Extended Kalman Filters [51], where mixtures, or sums, of Gaussians are employed.

An alternative approach is given by nonparametric filters. Here the belief probability distribution does not have an analytic, parametric, representation, and it is not represented by means of its moments or other statistics. A well known example are the particle filters, introduced by Gordon et Al. in 1993 [35], even though hints of the approach were already present in the literature of the fifties. These filters represent the belief distribution, in a Monte Carlo fashion, by means of a set of samples, called *particles*. The more particles are present in a certain region of the state space, the more likely these state will be. Such representation is approximate, yet, if enough particles are employed, it is able to represent multiple modes and, in the limit, any kind of probability distribution. Nevertheless, these Monte Carlo approaches suffer the *curse of dimensionality*; in other worlds, as the dimension of the state space increases, the number of particles needed to effectively represent a multivariate probability distribution over that space increases exponentially, undermining their applicability in high-dimensional state estimation problems.

These algorithms, along with a countless number of variations have been employed for state estimation in the past fifty years, and their widespread employment has not been limited to robotics. But what are the typical components of state spaces which we aim at estimating from sensor data?

2.2. State Spaces

Depending on the nature of the variables which are subject to estimation, different techniques have been employed and, as estimation problems become more and more challenging, limitations have been discovered in existing approaches, pushing the research towards different formulations of the state estimation problem.

Let us start with the very first case of pose tracking, i.e., the (recursive) estimate of the robot position and orientation with respect to a fixed reference frame, which is a common problem in mobile robot applications. In this case the state space is often composed by some parameterization of the transformation taking from the fixed to the moving reference frame (or vice versa). For this problem, EKFs were used since the sixties: a notable example is given by the application of the Kalman filter as a practical method for real-time onboard trajectory estimation and control in the Apollo program For an overview of other application of the Kalman filter in the aerospace field, see [36].

Here a digression would be needed on the notorious problem of the representation of rigid body transformations, even though the same discussion applies to many other situations. In these cases, state variables do not span over an Euclidean domain, instead they belong to *manifolds*, i.e., topological spaces that resemble the Euclidean one in a local neighborhood. Representing such variables often involves over-parametrization or integrity constraints. In general, preserving the non-Euclidean variables consistency after Bayesian updates has been an open issue for quite some time, at least in the robotic community, and a number of representation dependent techniques were developed, such as the use of normalization and Lagrange multipliers [82] with unit quaternions. Modern successful approaches employ Lie groups [38] and manifold encapsulation [43] to ensure that the state space is closed with respect to the Bayesian update operation.

Other types of state variables are often included in state estimation, such as sensor calibration parameters. Indeed, the measurement distribution is often conditioned on other quantities, i.e., $p(z_t|x_t, \xi)$, which might, or might not, in turn depend on time. This dependency, if dropped, might result in severe loss in estimation accuracy. A remarkable example consists in the biases of gyroscope and accelerometer sensors, which are not constant, differ from sensor to sensor and depend on motion, temperature, and a number of other factors. Indeed, if the angular velocity reading is biased, and the bias is substantial, and might change with time, we have to estimate is value *online*, in order to relate gyroscope readings to the robot state. Multiple EFK and UKF formu-

lation have been proposed in the literature which track for one or both of these parameters, for instance see [82]. The Bayesian filtering approach has also been employed in many other online parameter tracking problems, such as the systematic and non-systematic components of the odometry error [61], or the GPS latency [13].

Finally, a further example of state spaces is given by the problem of Simultaneous Localization and Mapping (SLAM), in which the map of the world is encoded by the positions of a set of landmarks, and a belief distribution is maintained and updated over the robot pose *and* the landmarks. This idea is generally attributed to Randall C. Smith and Peter Cheesemanin, in their work of 1987 [72], even though other pioneering work was done by Hough F. Durrant-Whyte and John Leonard in the early 1990s [56]. First approaches to the solution of the SLAM problem were based on EKFs and, in the monocular case, employed specific parameterizations, such ash the *unified inverse depth* [65], to represent landmark positions, so that a Gaussian distribution defined over that parametrization would better model the actual, non-Gaussian, uncertainty with respect to the landmark 3-D position [21].

A notable feature in SLAM is that the state dimension is not constant, and in principle unbounded. Indeed, the set of landmarks is not known a-priori: the goal in this problem is to map an unknown environment, while simultaneously localizing with respect to the map we are building. In general, as relevant features are detected, they augment the state space for which the robot is maintaining a belief distribution. However, the computational cost of the basic EKF increases with quadratic complexity in the number of landmarks, preventing real-time performance as the dimension of the map becomes significantly large. Moreover, during the Kalman filter update, while computing the new belief distribution the old state is marginalized. This in general introduces cross-correlations between each component of the state, resulting in a dense covariance matrix. Although the latter issue has been addressed employing Information Filters, e.g., in [79], other limitations of the filtering approach in SLAM pushed the research towards new formulations of the state estimation problem, capable of both explicitly representing problem sparsity and bounding the computational complexity.

2.3. Graph-based State Estimation

In 1997, a graph-based approach to the SLAM problem was proposed in [60]. In Lu and Milios formulation, nodes in the graph represent poses and landmark parameterizations (see Figure 2.1). If a landmark is visible

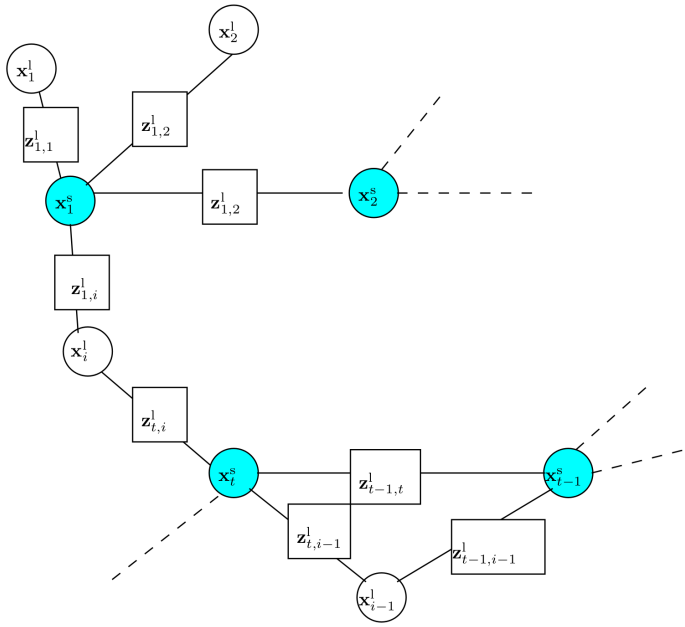


Figure 2.1.: A example of a graph formulation for the SLAM problem, where x^l_i are the landmark nodes, x^s_t are camera poses, $z_{t-1,t}$ are odometry measurements and $z^l_{t,i}$ are landmark observations, when landmark i is observed from pose t (The image was taken from the g^{2}o documentation).

from a certain pose, then an edge is added between the two. The state estimation problem is then formulated as a max-likelihood optimization over this graph in which the goal is to find the configuration of robot poses and landmarks such that the joint likelihood of all the observations is maximum. This ultimately requires to solve a large non-linear, least-squares, optimization problem. This approach appeared long ago in the photogrammetry and geodesy literatures and it is referred as *bundle adjustment*. However, many of the available results are little known in the vision and robotics communities, where they are gradually being reinvented. Graph-based approaches are nowadays considered superior to conventional EKF solutions [75], although major advancements in sparse linear algebra (see for example [29]) were required to make them competitive from the point of view of computational complexity. For a modern synthesis of these methods please see [80].

The advent of graph techniques in SLAM slightly changed the perspective from the state estimation point of view: while filters typically model state estimation as a recursive process performed measurement-

per-measurement and the state consists of the latest robot pose and all the landmarks, graph-based approaches attempt to estimate the full robot trajectory, and thus a (long) sequence of robot poses together with the landmark positions form the whole set of measurements. This notion was already present in *Kalman fixed-lag smoothers* [66], where the latest n poses are kept in the state.

Recently, the graph approach was generalized even further, now considering hypergraphs, called *factor-graphs* [52], where an edge, i.e., a factor, is incident to an arbitrary number of nodes. This leaded to a very powerful tool for multi-sensor fusion in general. Indeed, now sensor readings can be related to an arbitrary number of poses, landmarks, and other nodes, allowing to formulate complex observation models while maintaining a explicit representation of the problem sparsity. This formalism is employed also in other fields, as for example in telecommunications as a tool to decode turbo codes.

As an example of such generalization, we cite the *all source positioning and navigation (ASPN)* project [28], founded by DARPA, which aims at weakening the dependency on the GPS system for global localization by designing: (i) better inertial measurement units (IMUs) that require fewer external position fixes, (ii) alternate sources to GPS for those external position fixes and (iii) new algorithms and architectures for rapidly reconfiguring a navigation system with new and non-traditional sensors for a particular mission. Within this project, contemporary with respect to this work, sensor fusion methods have been developed that, while differing substantially from the implementation point of view, address the pose tracking problem in a similar way with respect to this work [49] [47]. In particular, different type of smart-factors [16] relate a set of robot poses with measurements depending on the generating sensor type, in a similar way with respect to our *logical sensors*. Moreover, as we do, many different information sources are considered, such as IMUs, odometry, GPS, and vision measurements.

2.4. Sensor Self-Calibration

As we have mentioned before, measurement distributions might depend on other sensor-specific quantities, which are often referred as sensor (calibration) parameters. There are countless examples of such quantities and the more measurement models become complex, the more parameters these model will exhibit. We have already cited biases and gains in accelerometers and gyroscope sensors. Here we add magnetometer distortions coefficients due to ferromagnetic properties of the

sensor placement surroundings, intrinsic camera calibration parameters and lens distortion coefficients, stereo camera rig baseline, relative alignment and placement of two sensors, kinematic parameters such as wheel radii, cornering stiffness, communication latencies, and so on. These parameters can be divided between those which change over time, and thus have to be part of the robot state and be tracked online, and those which do not change and can be determined once for all, such as geometric placement parameters or intrinsic camera calibration matrices.

Sensor calibration, even in the offline case, is still an open problem. Numeric and observability issues arise and it is often difficult to give proofs of convergence, or specify which are the requirements in the general case. Several ad-hoc methods have been proposed to handle very specific situations. The topic is still subject of active investigation; without any claim of completeness, we report [18], where kinematic parameters for a differential drive robot and the relative placement of a 2-D laser range-finder are estimated by means of a constrained linear least-squares formulation, a magnetometer hard and soft iron distortion calibration algorithm [83], and very recent works on Inertial Measurement Unit [77] and multiple 2-D lidar systems [41].

While most the mentioned approaches are well established in the literature, such as for the hand-eye calibration problem [45], which typically refers to determine the transformation between the robot end effector and a camera mounted on it, interest is growing for methods which attempt to remove specific assumptions, or environment structure, such as checkerboards in intrinsic camera calibration algorithms [20]. Indeed, research effort is pointing towards solutions in which a sensor, or a set of sensors, is able to autonomously calibrate its critical parameters, without the need for further information, such as the position ground truth obtained by means of an external motion capture system.

In 2011, Rainer Kümmerle, Giorgio Grisetti and Wolfram Burgard proposed a further extension to the SLAM problem, i.e., *simultaneous localization, mapping and calibration*, again building on the powerful factor-graph representation of the information fusion problem [55] [53]. In this inspiring work, the goal is not only to estimate the robot poses and the map of the environment, but also determine the values for other parameters appearing in the expressions of the measurement factors, employing only the reading from the sensors that are simultaneously undergoing calibration. In the cited work, the authors consider the displacement and misalignment of a 2D laser range-finder and the kinematic parameters of a differential drive robot, i.e., the wheel radii and the robot baseline. Even though they considered a specific case, already addressed by many other ad-hoc techniques, a general approach for sen-

sor self-calibration was proposed which ultimately forms the basis for this work.

2.5. General Frameworks

While estimation techniques begin to consolidate, position tracking and mapping capabilities remain critical for the success of most of the autonomous, mobile robot applications. Furthermore, parameter calibration is still an open issue in the general case and becomes more and more critical as robotic systems and perception architectures grow in complexity. Not to mention the fact that each robot has its own specifications, such as the available sensors, the required degree of accuracy in pose tracking and mapping, indoor or outdoor operating condition, 2-D or 3-D world assumptions, and so on. Thus robotic system developers, or researchers, can seldom deploy available solutions and often they have to enhance, adapt, tweak, or develop from scratch a pose tracking and/or mapping algorithm that fits specifications.

The advent of general solvers for non-linear, least-squares, optimization problems over factor graphs, such as g^2o [54], on which this work is based, GTSAM [30] and the ceres-solver [3] are enabling the development of robot and sensor independent, flexible and modular pose tracking, mapping and parameter calibration framework. What we would like is a set of mathematical techniques and software components sufficiently general so that they can be *configured*, and not adapted, to be deployed on very different robotic system, delivering *out-of-the-box* pose tracking, mapping and sensor parameter self-calibration.

This work aims at making a step towards this direction, providing a convenient environment to perform sensor fusion in mobile robotics, focusing on pose tracking and parameter self-calibration. Few similar solutions have been presented in the literature, such as the aforesaid ASPN project by DARPA. However, the ASPN framework does not attempt to estimate calibration parameters and also employs smart-factor capabilities to exclude landmarks from the optimization. Moreover no code has been released due to military restrictions. As for freely available software, we note the ETH ASL sensor fusion package [84], which address the problem of visual-aided, inertial, pose estimation and parameter self-calibration in micro aerial vehicles (MAVs) and it is based on a Kalman smoother, with very small lag window, as to allow delayed or out-of-sequence readings.

Chapter 3

Framework Overview

ROAMFREE is a software framework designed to provide off-the-shelf, modular and flexible pose tracking and sensor parameter self-calibration in mobile robotic applications. The main goals of the project include ensuring that the resulting software framework can be employed on very different robotic platforms and hardware sensor configurations and that it can be easily tuned to specific user needs by replacing or extending its main components.

To achieve these goals, critical choices have been made in terms of software engineering and in the formulation of sensor models and the information fusion problem. Indeed, ROAMFREE is divided into components, such as the problem handler, the sensor models and the solver algorithms, which hide their internal details under abstract interfaces.

In this chapter we discuss these choices in detail while giving an overall description of the software framework. We start from a high level component and functional description and we move down to the sensor models and state variables class hierarchy that ultimately allow to achieve the modularity and the reusability goals. Unfortunately, due to their intimate relations in the development of the framework, this overall description has to necessarily cover both software engineering concepts such as abstract interfaces and class hierarchy and probabilistic formulation of the information fusion problem. We will try to keep the discussion as organic as possible, and, where its clarity is not affected,

we will postpone mathematical details to Chapter 4.

3.1. Introduction

ROAMFREE is a flexible and modular framework designed to deliver to mobile robots and unmanned vehicles developers (i) off-the-shelf position and attitude tracking, (ii) intrinsic, extrinsic, and kinematic parameters self-calibration capabilities.

The information fusion problem is formulated as a fixed-lag smoother whose goal is to track not only the most recent pose, but all the positions and attitudes of the mobile robot in a fixed time window: short lags allow for real time pose tracking, still enhancing robustness with respect to measurement outliers; longer lags instead allow for offline calibration, where the goal is to refine the available estimate of sensor parameters. Hybrid scenarios are also considered in which a limited number of calibration parameters, such as the time-varying bias of a gyroscope sensor, can be tracked online, along with robot poses.

The core of ROAMFREE lies in a software module that keeps and updates the probabilistic representation of the sensor fusion problem in terms of a *factor graph*, composed of pose and sensor parameter nodes and sensor error models connecting them. Other modules, such as the outlier rejection module, which detects and exclude incoherent sensor readings from the estimation, and inference algorithms performing state estimation, operate upon this representation.

The framework ships a set of high level sensor models which can be configured in terms of calibration parameters, and geometric displacement on the mobile robot, allowing the end user to precisely describe its robot perceiving architecture. The flexible and modular factor graph formulation of the sensor fusion problem allows to deal with an arbitrary number of multi-rate sensors, i.e., different sensors producing readings at different rates, having non-constant frequencies of operation, and possibly producing out-of-sequence data.

In the development of the library, we aim at delivering a software tool which is independent from the actual hardware machinery, or software algorithms, which originate the odometric information. ROAMFREE sensor models are logical descriptions of the actual sensors characterized in terms of measurement domain and geometric placement. We choose not to deal directly with hardware sensors, providing software interfaces for widespread commercial devices, on the contrary, we follow a *black box* approach focusing on the nature of the information sources. Indeed, we provide error models that allow to handle all the measurement domains

commonly employed in pose tracking.

As an example, consider a gyroscope being part of an inertial measurement unit and a visual odometry algorithm processing images acquired by a calibrated camera: both information sources can be seen as *logical* angular velocity sensors. As long as the sensor abstract model is expressive enough, from a pose tracking point of view there is no need to distinguish between these two information sources. Indeed, what we need is just a parametric error model which can be configured by the user to handle the peculiarities of the actual sensor employed (e.g., bias in case of a gyro and unknown scale in monocular visual odometry algorithms). The idea of modeling sensors in an abstract way has been first proposed by Tom Henderson in [42].

Another key feature of the framework lies in the modularity of the implementation: mathematical and software engineering techniques have been employed such that the main framework components, the logical sensors models, the state variable representations, the sensor fusion problem handler and the solver algorithms hide their internal details under abstract interfaces. This allow the end user to instantiate the framework with one or another implementation of these components in a transparent way, choosing the one that best fits its application needs.

Consider for instance the representation of 3-DoF rotations, for which several choices exist; each choice exhibiting its own advantages and disadvantages (e.g., Euler angles, quaternions, 3×3 rotation matrices, etc.). When formulating a magnetometer error model, i.e., an equation which relates the current orientation of the sensor with its current reading of the Earth magnetic field, we do not need to know the internals of the sensor orientation representation; we just require it to expose three operators: (i) the composition of two rotations, (ii) the inverse, (iii) the application of the rotation to a real vector. At the same time, even the sensor fusion algorithm can be designed to ignore these details and to work with arbitrary state variable representations by relying on operators to update its value, ensuring that possible constraints induced by over-parametrization are satisfied.

In the next sections we provide an overview of the main concepts and techniques that form the basis of this work. We first discuss the information fusion problem formulation and give a high level description of the solution techniques, next we introduce the sensor model hierarchy, from abstract sensors, which describe an arbitrary information source placed on the mobile robot, to logical sensors, which add a sensor dependent measurement model allowing to relate readings to robot poses and calibration parameters. Finally we give a high level description of the framework main components and modules.

3.2. Core Fusion Engine

In ROAMFREE, we represent the joint probability of sensor readings given the current estimate of the state variables, i.e., the robot poses and the sensor calibration parameters by means of a factor graph [52]. This is a graphical model and expresses the factorization of a function of several variables, e.g., a joint probability density function, into the product of *factors* which involve only a, usually small, subset of the variables, allowing to explicitly model the sparsity of the problem.

In our case, the nodes in the factor graph contain the state variables of the problem, which are all the robot poses in a given time window, and the sensor calibration parameters, such as gains, biases, displacements or misalignments. Factors represent measurement constraints: each factor is an hyperedge and connects multiple pose and calibration parameters nodes, and encodes a non-linear error model which depend on the type of the logical sensor:

$$e_i(t) = \hat{z}(x) - z + \eta \tag{3.1}$$

where $\hat{z}(x)$ is a measurement predictor computed as a function of the incident nodes, z is the actual sensor reading and it is directly associated to the corresponding factor, finally η is a zero-mean, Gaussian, noise encoding measurement uncertainty. Equation 3.1 yields the difference between the expected sensor reading given the robot state and calibration parameters at time t, and the actual measurement produced by the sensor. In Figure 3.1 it is possible to see a simplified instance of the factor graphs considered.

The advantages of the factor graph formulation for the pose and parameter tracking problem are many:

- it allows for an arbitrary number of sensors to be handled in a modular and independent way. Indeed, different sensor models implement the abstract hyperedge interface and they are handled uniformly as they are inserted into the graph.

- Sensors can be dynamically turned on and off online. Factors are inserted into the graph asynchronously as new readings are available for a certain information source.

- Out-of-order measurements, i.e., sensor readings which are received in a wrong order with respect to their timestamps, can be handled in a natural way simply picking the pose nodes with appropriate timestamps when constructing the corresponding factor. In a similar way, it is possible to deal with arbitrary and

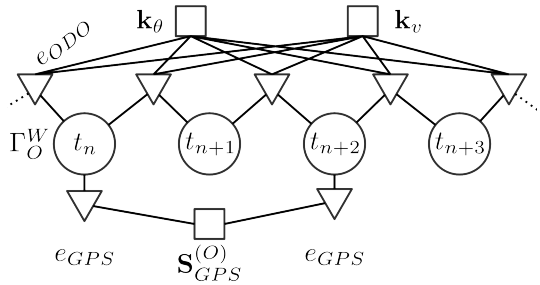


Figure 3.1.: An instance of the pose tracking factor graph with four pose vertices $\Gamma_O^W(t)$ (circles), odometry edges e_{ODO} (triangles), two shared calibration parameters vertices $\mathbf{k}_v$ and $\mathbf{k}_\theta$ (squares), two GPS edges e_{GPS} and the GPS displacement parameter $\mathbf{S}_{GPS}^{(O)}$.

non-constant reading rates. Note that, even if new factors constraint past nodes, they still contribute to the refinement of the most recent pose estimate through the other constraints already present in the graph.

- From the estimation quality point of view, it has been argued that these kind of systems are more accurate, and, in certain circumstances, even faster than traditional filters, such as EKFs [75].

- The high degree of sparsity of the considered information fusion problem is explicitly represented and can be exploited by inference algorithms. Indeed in our case a factor may involve up to three robot poses (see Section 4.2); moreover, it is difficult to imagine a robot employing much more than ten sensors, implying that each pose is incident to a limited number of factors.

The information fusion problem is solved maximizing the likelihood of the sensor readings with respect to the state variables. This is a non-linear, weighted, least-squares optimization problem and a numerical solution is found by means of the popular Gauss-Newton or Levenberg-Marquardt [34] algorithms.

3.2.1. Factor Graph Construction

In ROAMFREE, a factor graph encodes the probability density of sensor readings given the current estimates of the state variables. In this hypergraph each node represents a robot pose at some time t or a sensor

parameter, and each hyperedge corresponds to a measurement constraint involving one or more pose nodes and calibration parameters.

Whenever a new sensor reading is available, the Graph Management component is responsible for updating this representation and instantiating a new factor, according to the type of the information source, and inserting it into the graph as it is incident to the pose and sensor parameter nodes required to evaluate the measurement likelihood (i.e., the sensor error function). In particular, the measurement domains, i.e., linear velocity, acceleration, and so on, specify how many pose nodes have to be associated to the factor, and which are the sensor calibration parameters involved in the error function.

For instance, consider an odometry measurement obtained at time t from the wheel speed readings in a differential drive robot. This information defines a hyperedge involving the robot poses at time $t-1$ and t, the wheels radii and baseline distance parameters. This edge constraints the difference of the two poses according to the forward kinematics of the robot. In Figure 3.1 it is possible to see a more complex example involving odometry edges, e_{ODO}, which, in turn, depend on two kinematics parameters, $\mathbf{k}_\theta$ and $\mathbf{k}_v$, and GPS edges, e_{GPS}, constraining the position of the robot frame up to a misplacement parameter, $\mathbf{S}_{GPS}^{(O)}$.

In case the sensor reading is newer with respect to the latest pose available in the graph, a new pose node has to be instantiated. Since the sensor fusion problem is formulated as a non-linear optimization, an initial guess is needed for the new state variable. If the sensor reading provides enough information, a prediction for the new robot pose can be obtained based on the latest available estimate and on the sensor reading itself. Otherwise, the measurement handling is delayed until such information becomes available. Moreover, as new pose nodes are inserted into the graph, old ones have to be removed so that the length of the fixed-lag window remains constant. This causes old factors to be removed as well, implying information loss. To avoid this, old nodes are marginalized and an new factor is inserted over their Markov blanket in a way such that it is equivalent to the lost edges, in the neighborhood of the current node estimates.

3.2.2. Least-Squares Optimization on Manifolds

In the RAOMFREE framework the joint probability of sensor readings given robot poses and calibration parameters is encoded in a factor graph by means of hyperedges that relate sensor readings to the current estimates of the state variables. In the following we discuss in details how the information fusion problem is formulated as a non-linear, weighted,

least-squares optimization and solved by means of Gauss-Newton, or Levenberg-Marquardt algorithms. Moreover, since the variables for this optimization problem span over non-Euclidean manifolds, such as the special Euclidean group $SE(3)$, which encodes rigid body 6-DoF transformations, these algorithms are formulated such that they operate consistently and independently with respect to the variable domains.

Let $e_i(x_i, \eta)$ be an error function as in Equation 3.1, associated to the i-th edge in the hypergraph, where x_i is a vector containing the variables appearing in any of the nodes connected by the hyperedge and η is a zero-mean Gaussian noise vector. Thus e_i is a random vector and its expected value is approximated as $\overline{e}_i(x_i) = e_i(x_i, \eta)|_{\eta=0}$. Since e_i can involve non-linear dependencies with respect to the noise, its covariance Σ_η is computed by means of linearization, i.e.:

$$\Sigma_{e_i} = J_{i,\eta} \Sigma_\eta J_{i,\eta}^T \big|_{x_i = \breve{x}_i, \eta=0} \tag{3.2}$$

where $J_{i,\eta}$ is the Jacobian of e_i with respect to η evaluated in $\eta = 0$ and in the current estimate $\breve{x}_i$. Here some technicalities arises in the definition of $J_{i,\cdot}$ that we will discuss in detail in Section 4.5.

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

$$\mathcal{L}_i(x_i) = \overline{e}_i(x_i) \Omega_{e_i} \overline{e}_i(x_i), \tag{3.3}$$

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

$$\mathcal{P}: \quad \arg\min_x \sum_{i=1}^{N} \mathcal{L}_i(x_i). \tag{3.4}$$

It is possible to see that this is a non-linear least-squares problem where the weights are the information matrices associated to each factor. If a reasonable initial guess for x is known, a numerical solution for $\mathcal{P}$ can be found by means of the popular Gauss-Newton (GN) or Levenberg-Marquardt (LM) algorithms, see for example [34].

In GN, the error functions are approximated with their first order Taylor expansion around the current estimate $\breve{x}$:

$$e_i(\breve{x} + \Delta x) \simeq e_i(\breve{x}) + J_{i,\Delta x} \Delta x \tag{3.5}$$

where $J_{i,\Delta x}$ is the Jacobian of e_i with respect to Δx, evaluated at $\Delta x = 0$. Substituting (3.5) in (3.4) yields a quadratic form which can be solved

in Δx. Then $\breve{x}$ is replaced with $\breve{x} + \Delta x$ and Ω_{e_i} is updated for each edge as in (3.2). These steps are repeated till termination criteria are met.

However, the vector x_i may contain variables which span over non-Euclidean space. This is very relevant with respect to our case: indeed, robot poses belong to the special Euclidean group $SE(3)$. Here over-parametrized representations are often employed, such as unit quaternions or 4×4 matrices, and the usual formulations of the GN algorithm could fail to preserve the constraints induced by over-parametrization. We address this issue by means of *manifold encapsulation* [43].

In Equation 3.5 the operator $+$ is replaced with the operator $\boxplus$, which generalizes the sum operation for non-Euclidean domains, as we will discuss in details in Section 4.1.2. The x manifold has been *encapsulated* in the sense that the GN algorithm can access the internal representation of x only by means of the $\boxplus$ operator, which, depending on the variable type, consistently maps a local variation Δx in an Euclidean space to a variation on the original manifold. Note that substituting $\boxplus$ with regular $+$ in (3.5) would yield solutions for Δx which could break the constraints induced by over-parametrization.

From the discussion above, it is possible to see that solver algorithm rely only on the following operations: (i) the evaluation of the error functions, (ii) the computation of the Jacobians of the error functions with respect to state variables increments Δx, (iii) the $\boxplus$ operator, that allow to update the state variables estimate without having to deal with the constraints induced by over-parametrization. As we will discuss in next sections, these operations are made available through abstract interfaces, achieving the decoupling between state variables representation, sensor models and estimation algorithms.

As for the implementations of the least-squares solvers we rely on the g^{2}o [54] software library, a general framework for graph optimization, fine tuned to exploit the sparsity of factor graph optimization problems such as the one considered in this work. At the present stage of development, three solver algorithms are available: Gauss-Newton, Levenberg-Marquardt and Peconditioned Conjugate Gradients. Nevertheless, many sensor fusion algorithms, such as an Extended Kalman Filter, or incremental smoothing, such as iSAM2 [49], can be formulated such that they rely only on the available primitives and implemented as new solvers without changes in the rest of the architecture.

3.3. Sensor modeling

In the ROAMFREE project we aim at the development of a general pose tracking and sensor calibration framework that is independent from the actual robotic platform, sensors, and middleware used for robot development. These goals required the design of generic models for odometry sensors that abstract as much as possible from the nature of the information sources and the fusion engine. To this extent, due to the wide variety of sensors and algorithms available in motion tracking, we decided not to base our models for pose tracking and sensor calibration on *physical sensors*, i.e., sensors hardware and corresponding processing, but on *logical sensors* described in terms of the type of measurements they produce. This shift, from the physical process, and processing, to the intrinsic type of information contained in the data allows us to work at a higher level, providing more flexibility and modularity, without missing any detail required to perform accurate odometry fusion tasks. In the following we highlight the hierarchical structure of the sensor models (see Figure 3.2 for reference).

As we have briefly mentioned in the previous section, sensor readings compose measurement constraints as edges in the factor graph representing the probabilistic formulation of the information fusion problem. At the graph level, we find the declaration of the operations needed by solver algorithms. These refer to the evaluation of: (i) the edge likelihood function given the current estimates of incident nodes (e.g., robot poses and sensor calibration parameters), (ii) the first order derivative of the edge error models as a function of the incident variables nodes, or, in other words, their direction of steepest descent. Note that, while the internal of these operations are clearly dependent on the error model implemented in the considered edge, this outer interface allows to treat all the different edges in an uniform way.

The next level in the hierarchy consists of *abstract sensors*. This level describes the geometric properties of the actual sensor placement on the mobile robot. Due to displacement and misalignment between the robot and the sensor reference frames, kinematic properties such as velocities and accelerations may differ between the two. Abstract sensors evaluate such difference as a function of geometric calibration parameters and compute an extended kinematic state for the sensor frame by means of successive poses finite differences, as we will discuss in detail in Section 4.2. Building upon this, *logical sensors* extend these classes providing error models for the actual sensor measurements. More precisely, a measurement predictor, which is function of the kinematic properties of the sensor reference frame, is evaluated and compared with

the actual sensor reading, providing a measure of the likelihood of the robot state estimate with respect to the sensor reading considered. This two-level class hierarchy achieves the decoupling between the equations required to handle misaligned and misplaced hardware sensors with the ones employed to actually model the information domains, easing the development of new sensor models and allowing to test different kinematic predictor formulations with the same error models.

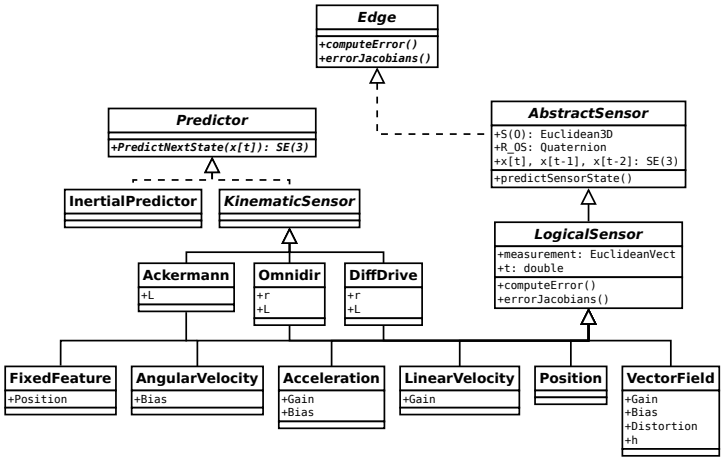


Figure 3.2.: Outline of the sensor model class hierarchy.

3.3.1. Abstract Sensors

The top level of the sensor hierarchy consists in abstract sensors, which give a geometric characterization of information sources. Indeed, the ideal sensor placement in which all sensors positions and orientations match the ones of the odometric center of the mobile robot is usually unachievable, or impractical (See Figure 3.3). To handle this, we characterize each (abstract) sensor S_i by means of two geometric parameters, i.e., $\mathbf{S}_i^{(O)}$, the origin of the i-th sensor reference frame with respect to the odometric frame O, and $\mathbf{R}_{S_i}^{O}$, the rotation taking from O to S_i. These two parameters yield a transformation $\Gamma_{S_i}^{O} = [\mathbf{S}_i^{(O)}, \mathbf{R}_{S_i}^{O}]$ which expresses the i-th sensor reference frame with respect to the robot odometric center O. Note that our goal is to track the transformation taking from the world fixed frame W to the mobile robot frame O.

The characterization of the sensor geometric placement on the robot is crucial for accurate pose tracking, and anyhow at least a rough guess for the $\mathbf{S}_i^{(O)}$ and $\mathbf{R}_{S_i}^{O}$ calibration parameters is required to perform the fusion of multiple sensor information properly. To see why, consider

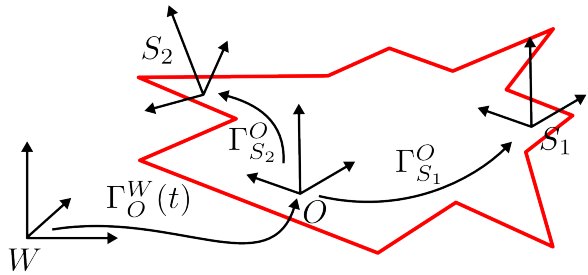


Figure 3.3.: Reference frames employed in the ROAMFREE sensor fusion library. Abstract sensors S_1 and S_2 are misplaced and misaligned with respect to the Odometric reference frame O.

for instance a laser range-finder placed at the front-right corner of a differential drive robot. If a scan-matching algorithm is employed to process its point cloud output, it is possible to obtain an estimate of the sensor linear and angular velocities, which may be inconsistent with the one obtained by means of forward kinematic, e.g., when the robot is rotating in-place along its z axis: in this case the range-finder would report a non-zero linear velocity estimate, since, due to its displacement with respect to the odometric center, the range-finder undergoes an additional translation with respect to the fixed world.

Geometric placement parameters are often determined by means of direct inspection on the robot. However, this can be rather difficult, as for 3-DoF rotations, and, in any case, impractical, as the sensors may be mounted in position which are difficult to reach, or even the kinematic reference frame O might be difficult to identify. These situations are handled by means of the ROAMFREE sensor self-calibration capabilities, as we will demonstrate in Chapters 6 and 7.

Given the state of the robot, i.e., the position and orientation of the odometric reference frame with respect to the world, by means of the geometric parameters available at the abstract sensor level, we can estimate the kinematic properties of the S_i reference frame. More precisely, given the current estimates of the robot pose, $\Gamma^W_O(t)$, $\Gamma^W_O(t-1)$ and $\Gamma^W_O(t-2)$, and the sensor placement parameters, $\mathbf{S}_i^{(O)}$ and $\mathbf{R}^O_{S_i}$, we can derive an extended state for the sensor frame S_i, namely, $\hat{x}_{S_i}(t)$, which is composed by $[\hat{S}_i^{(W)}(t), \hat{R}^W_{S_i}(t)]$, the position and orientation of the sensor with respect to the world frame, $[\hat{v}^{(S_i)}(t), \hat{\omega}^{(S_i)}(t)]$, its linear and angular velocities expressed in the local frame, and $[\hat{a}^{(S_i)}(t), \hat{\alpha}^{(S_i)}(t)]$, the linear and angular accelerations. These quantities characterize the motion of

the sensor reference frame with respect to the world. To derive $\hat{x}_{S_i}(t)$, we rely on a discrete-time formulation of the 6-DoF rigid body motion equations. These predictors are employed, in a hierarchical fashion, to build the measurement domain models. The mathematical details involved in the computation of $\hat{x}_{S_i}(t)$ are discussed in Section 4.2.

3.3.2. Logical Sensors

The aim of logical sensors is to provide a predictor, $\hat{z}(t)$ for the expected sensor readings as a function of the sensor kinematic state, which is a property of the abstract sensors, so to allow the definition of an error function of the form

$$e(t) = \hat{z}\big(t; \hat{x}_{S_i}(t), \xi\big) - z(t) + \eta, \tag{3.6}$$

where η is a zero-mean, Gaussian, noise vector and $e(t) \in \mathbb{R}^n$ and ξ is a vector or sensor-specific calibration parameters.

The specific form of the measurement predictor $\hat{z}$ depends on the type of measurement we are considering, thus we have to specialize the concept of logical sensor to handle the specific measurement domains: (i) absolute position and/or orientation, (ii) linear and angular velocity in sensor frame, (iii) acceleration in sensor frame, (iv) vector field in sensor frame, (v) landmark pose with respect to sensor. In this sense, a logical sensor is a black box source of information characterized by its measurement domain and inherits from the abstract sensor the geometric properties which specify its displacement with respect to the robot odometric reference frame. Note that the measurement domains mentioned above allow to handle all the hardware sensors and software algorithms commonly employed in mobile robotics pose tracking, e.g. GPS, SLAM and Visual Odometry algorithms, gyroscopes, accelerometers, magnetometers, etc. Nevertheless, different models can be easily introduced defining further implementations of the logical sensor interface.

In Equation 3.6 we have enlighten the dependency of the measurement predictor $\hat{z}(t)$ on the kinematic state of the sensor frame and on further, sensor specific, calibration parameters, ξ. Equation 3.6 indirectly relates the robot states $\Gamma_O^W(t)$, $\Gamma_O^W(t-1)$ and $\Gamma_O^W(t-2)$ with the sensor reading $z(t)$ through the sensor extended kinematic state $\hat{x}_{S_i}(t)$. It is possible to see that zero-mean error $e(t)$ is obtained when the prediction of the sensor reading matches the actual one. Note that $z(t)$ seldom gives full information on the robot state and, even if this was the case, it is difficult to invert the sensor model to give a closed form expressions for $\Gamma_O^W(t)$ as a function of $z(t)$. Yet, as we have discussed in the previous section, an estimate for the state variables can be obtained jointly minimizing

all the error functions $e(t)$ with respect to all the robot poses, provided that at least a rough initial guess is available.

Each logical sensor implementation introduces a different set of parameters ξ to model domain specific sources of distortion, bias or other quantities which have to be known to evaluate the predictor in Equation 3.6. These parameters can be enabled or disabled by the user to accommodate for specific properties of the information source considered. For instance, gyroscope error models reported in the literature take into account a time-varying bias, related to the nature of the underlying physical process. In our case, an angular velocity logical sensor would be employed, enabling its bias correction parameter. A more complex example is the error model employed in the vector field logical sensor when dealing, for instance, with a magnetometer:

$$e(t) = \overbrace{\mathbf{A}\left(\hat{R}_S^W(t;\mathbf{R}_S^O)\right)^{-1}\vec{\mathbf{h}}^{(W)} + \mathbf{b}}^{\hat{z}(t)} - z(t) + \eta, \tag{3.7}$$

where a bold font highlights sensor calibration parameters. Here we have made explicit the sensor orientation predictor $\hat{R}_S^W(t)$ dependency on the sensor misalignment parameter $\mathbf{R}_S^O$ (see Section 3.3.1). In case this logical sensor is employed to handle magnetometer readings, the distortion matrix $\mathbf{A}$ and the bias vector $\mathbf{b}$ are enabled and account for hard and soft iron distortion effects [83], while $\vec{\mathbf{h}}^{(W)}$ has to be set according to the local value of the Earth magnetic field, which in this case is assumed to be constant in the robot operation area.

The ROAMFREE sensor library includes full parametrized implementations of a wide variety of error models which can be employed to model most of the physical sensors and processing algorithms commonly employed in mobile robot pose tracking. Moreover, the hierarchical structure of the sensor models allows the end user to easily extend or refine the provided suite. The mathematical details for each available logical sensor are given in Section 4.3.

As a final remark, we briefly anticipate how the decoupling between abstract and logical sensor is achieved in the evaluation of Jacobians of the error functions with respect to the state variables. This operation is required to provide the solver algorithms with a notion of the error functions direction of steepest descent. For instance, it is employed by Gauss-Newton solvers to compute the linearized system Hessian matrix, and by Extended Kalman Filters to perform state and covariance updates. Here we have to compute the Jacobian matrix of the error function $e(t)$ with respect to the state variables involved, which consist in one or more robot poses, the sensor displacement and misalignment

parameters, $\mathbf{S}_i^{(O)}$ and $\mathbf{R}_{S_i}^O$, and any other parameter introduced by the logical sensor. Here we split this evaluation into two steps: first the logical sensors compute the Jacobian of $e(t)$ with respect to $\hat{x}_{S_i}(t)$. Next, the abstract sensor evaluates the Jacobian of $\hat{x}_{S_i}(t)$ with respect to the actual state variables $\Gamma_O^W(t)$, $\Gamma_O^W(t-1)$, $\Gamma_O^W(t-2)$, $\mathbf{S}_i^{(O)}$ and $\mathbf{R}_{S_i}^O$. As we remember from calculus, the required Jacobian matrix is given by the matrix product of the two blocks above. Regarding the other parameters which are introduced by the logical sensors, if any, e.g., $\mathbf{A}$ and $\mathbf{b}$ in Equation 3.7, the Jacobian matrix of the error function with respect to these variables can be computed directly at the logical sensor level. In this way, no notion of the actual error function is required at the abstract sensor level. Furthermore, the error function can be formulated, and its Jacobian matrix evaluated, without having to deal with the internal form of the predictor $\hat{x}_S$. See Section 4.5 for details.

3.3.3. Forward Kinematics Logical Sensors

A special class of logical sensors consists in kinematic models, e.g., differential drive, Ackermann, omnidirectional, and so on. Readings coming from this kind of logical sensors are more expressive then their more general counterpart since they directly characterize the robot motion. More precisely, both the linear and angular velocity of the odometric reference frame can be computed as a function of these sensor readings. Thus, a predictor of the next robot state $\hat{\Gamma}_O^W(t+1)$ can be constructed, given the current state $\Gamma_O^W(t)$ and $z(t)$. These logical sensors usually introduce kinematic parameters, such as wheel radius, baseline, number of encoder ticks per revolution, and so on, which are required to compute both the forward kinematic and the $\hat{z}(t)$ predictor.

As an example, consider a differential drive robot in which two encoders read wheel speed, ω_l and ω_r. The well known forward kinematic equations for such a robot read as:

$$\begin{aligned} \hat{v}_x^{(O)}(t) &= \tfrac{\mathbf{r}}{2}\big(\omega_r(t) + \omega_l(t)\big) \\ \hat{\omega}_z^{(O)}(t) &= \tfrac{\mathbf{r}}{\mathbf{L}}\big(\omega_r(t) - \omega_l(t)\big) \end{aligned} \tag{3.8}$$

where the $\mathbf{r}$ parameter is the wheel radius and $\mathbf{L}$ is the wheel baseline distance. Note that if we attempt to compute the full 6-DoF $v^{(O)}$ and $w^{(O)}$ from the encoder readings only, we have to implicitly assume that planar motion and no slippage occur. These assumptions constrain the remaining components of the linear and angular velocity in Equation 3.8. A simple Euler, or more complex Runge-Kutta, integration scheme yields the required predictor $\hat{\Gamma}_O^W(t+1)$.

Forward kinematics logical sensors allow the use of the full state predictor to compute reasonable initial guesses for the robot next state before the sensor fusion algorithm is started. Moreover, they often read quantities which are actively driven, i.e., the robot is *controlled* by means of the quantities these sensors measure. Thus, it is possible to employ the predictor above with the actuator setpoints $u(t)$, which will affect the observed quantities starting at time $t+1$, instead of sensor readings $z(t)$, which are related to control actions happened in the past. For instance, the differential drive robot in the example above certainly has a control loop which regulates the speed of the wheels to follow a known, and available, setpoint $u(t)$. When we have to compute an initial guess for the state Γ_O^W at the time $t+1$ the encoder readings $z(t+1)$ are not available yet, so we do not know the *actual* wheel speed. However, the last control setpoint $u(t)$ is available and it affects the system starting from time $t+1$. Thus, under reasonable assumptions, it can be employed in place of $z(t+1)$ to evaluate the Forward Kinematic predictor.

3.4. State Variables

As it has been already introduced, ROAMFREE provides modularity of state and parameters representation too; in this section we discuss the hierarchy of *state variables*, which fill factor graph nodes and include both 6-DoF robot poses and sensor calibration or geometric parameters.

Each state variable has its own domain, eventually non-Euclidean; in this work a technique called *manifold encaplsulation* [43] is employed: it allows to handle variables whose domain is a manifold, i.e., a topological space in which each point has a neighborhood that *resembles* the Euclidean space, in a transparent way, meaning that that the sensor fusion algorithm and the sensor models do not need to know the particular, non-Euclidean, structure of the space they are operating upon, nor they have to access variables internal representation, nor they have to take any special care to ensure its consistency. Indeed, they rely only on operators which define the state variable interface. Hiding the internal representation of state variables achieves the decoupling of sensor fusion algorithms and sensor model formulations from the actual state variable representation. We will discuss the ROAMFREE implementation based on Hamiltonian unit quaternions in Section 4.1.2.

Besides their domain, state variables can be *fixed* or not[1]. A fixed variable is treated like a constant, i.e., its value is considered to be known

[1] We remark that the term *fixed* does not have to be confused with the time-invariant term. The difference will become clear in the following of this section.

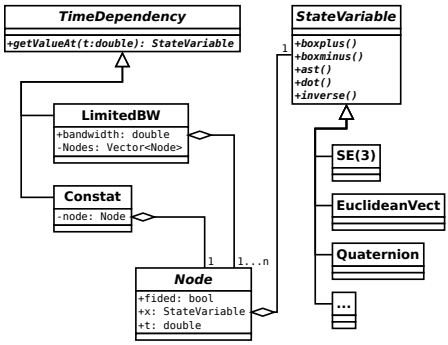


Figure 3.4.: The state variables class hierarchy.

and it is not subject to estimation during the filtering process. Vice versa, if this property is set to `false`, the sensor fusion engine tries to estimate its value. This property can also be changed online. Consider, for instance, a state variable holding the differential drive kinematic parameters, $\mathbf{r}$ and $\mathbf{L}$; it is known that their observability depends on the robot trajectory [19]: user developed heuristic could monitor this condition and enable refinement of the kinematic parameters estimate only when enough information is available.

State variables are also characterized by their dependency on time. At the present stage of development, we consider *constant variables*, i.e., their value does not depend on time, or it is assumed to be constant along the time window considered, or *time-variant with limited bandwidth*. In the second case the user can specify the maximum frequency at which the variable parameter is supposed to change as a function of time. To compute the value of the parameter at time t we rely on a Lanczos resampling scheme [81] (see Section 3.4.2 for details).

3.4.1. Variable Domains

As previously introduced, each state variable has its own domain. Well known examples of variable domains which are not Euclidean are unit quaternions, often employed to represent 3-DoF rotation, and elements belonging to the space of 6-DoF rigid transformation, $SE(3)$. The representation of these variables is often overparametrized, i.e., it is composed by more variables with respect to the domain degrees of freedom, and involves constraints (e.g., the norm of unit quaternions must be 1, a rotation matrix must be orthonormal, and so on). State variables belonging to such domains cannot be correctly handled simply assuming they were Euclidean. A good lesson come from the use of unit quater-

nion in EKFs: during updates, unless special care is taken by means of *ad-hoc* methods, e.g., Lagrange multipliers [82], the norm of the quaternions eventually diverges from 1 and normalization has to be performed. Furthermore, due to overparametrization, the covariance matrix associated to the quaternion variable is always ill-conditioned.

In our work we follow the idea in [43] and define a state variable interface which requires a set of operators to be implemented. These allow both the error functions to be evaluated and the solver algorithms to perform state estimation without the need to know or to handle the internals of state variable representations. These operators are:

1. $\boxplus : \mathcal{M} \times \mathbb{R}^n \to \mathcal{M}$
2. $\boxminus : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^n$
3. $\cdot : \mathcal{M} \times \mathbb{R}^n \to \mathbb{R}^n$, default action on $\mathbb{R}^n$
4. ${}^{-1} : \mathcal{M} \to \mathcal{M}$, inverse
5. $\circ : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$, composition

The $\boxplus$ operator applies a local, Euclidean, increment to the non-Euclidean variable belonging to the manifold $\mathcal{M}$, and, as we introduced in Section 3.2.2, it is employed by solver algorithms to update state variables ensuring the consistency of their internal representation. The $\boxminus$ operator can be thought as an inverse of the previous operator, namely, $y \boxminus x$ gives the element $\delta \in \mathbb{R}^n$ such that $x \boxplus \delta = y$. The $\cdot$ operator performs the manifold default action on a real, Euclidean, vector, e.g., the application of a 3D rotation to an Euclidean vector rotates the vector. The other two operators return the variable inverse and combine two variables in the natural sense with respect to the variable domain. Note that there are a number of subtleties regarding the operators above. Refer to Section 4.1.2 and [43] for mathematical details.

3.4.2. Time Dependencies

In this section we describe choices for state variables time dependencies available in the ROAMFREE sensor fusion library. Let us start discussing an example.

Suppose we are employing a gyroscope to track the angular velocity of a mobile robot. It is known that these kind of sensors are biased. The

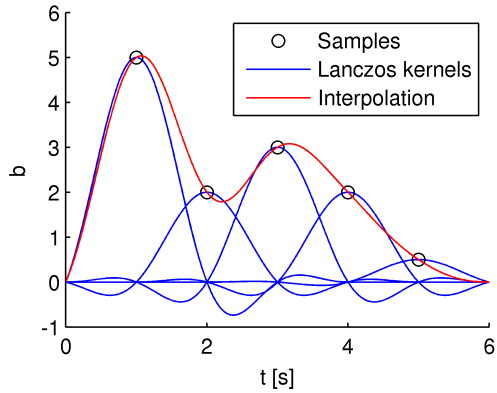


Figure 3.5.: Example of Lanczos resampling. Five samples are interpolated to produce a parameter signal whose maximum bandwidth is 0.5 Hz.

simplest error model in this case would be:

$$e(t) = \overbrace{\omega(t) + \mathbf{b}(t)}^{\hat{z}(t)} - z(t) + \eta, \tag{3.9}$$

in which we have assumed that the gyroscope observes the true angular velocity, up to a bias, ignoring any other error sources such as axes non-orthogonality. Since there is no way to directly observe $\mathbf{b}$, if we lack for redundant observations regarding $\omega(t)$, it is easy to see that the norm of $e(t)$ can be arbitrarily reduced by selecting proper values of $\mathbf{b}(t)$, thus compromising the information carried by $z(t)$. One way to address this issue is to assume that $\mathbf{b}(t)$ does not change with time, yielding the first type of time dependencies available in ROAMFREE, i.e., *constant* state variables. In this case the sensor fusion engine tries to estimate the unknown value of $\mathbf{b}(t)$ assuming that $\mathbf{b}(t_1) = \mathbf{b}(t_2)$, for each t_1 and t_2 in the active time window.

While the assumption above may hold if we consider only local time windows, the gyroscope bias is known to be time-varying and thus a constant state variable would fail to model its behavior once a long enough time window is considered. A possible solution is to exploit the fact that $\mathbf{b}(t)$ is known to change *slowly*. This introduces the second type of time-dependencies currently available: *limited bandwidth* state variables. In particular we let the user choose the maximum bandwidth f at which variable is supposed to change over time, thus introducing a constraint on the values of $\mathbf{b}(t)$.

From signal processing theory we know that if we convolve a discrete

time signal with sampling frequency $f_c = 2f$ with the *sinc* function we obtain a continuous time signal with bandwidth f. Thus, a limited bandwidth parameter can be fully described by the set of samples $\mathbf{b}_k = \mathbf{b}(t)$, $t = k/f_c$, $k \in \mathbb{Z}$. To compute its value at an arbitrary time t we employ Lanczos resampling, i.e., we convolve the samples with the Lanczos kernel (see Figure 3.5):

$$L(t) = \begin{cases} a\frac{sin(2\pi ft)sin\frac{2\pi f}{a})}{(2\pi ft)^2} & if -a < 2\pi ft < a \\ 0 & otherwise \end{cases} \tag{3.10}$$

where the parameter a is an integer, typically 2 or 3. Unlike the *sinc* function, the Lanczos kernel has compact support, thus only a limited number of samples (i.e., $2a$ samples) contribute to determine the parameter value at time t, which in our example is given by:

$$\mathbf{b}(t) = \sum_{k=\lfloor 2ft \rfloor - a+1}^{\lfloor 2ft \rfloor + a} \mathbf{b}_k L(2\pi ft - k). \tag{3.11}$$

Note that this does not solve the issue discussed above in the general case. However, now it seems much more difficult to arbitrary reduce $||e(t)||$ in Equation 3.9 choosing $\mathbf{b}_k$, other than samples of the true bias.

Limited bandwidth state variables are useful in situations in which we have to track time-varying quantities such that their value cannot be assumed constant over the time window considered in the fixed lag smoother. A typical example is the calibration problem in which hundreds of seconds of sensor readings are considered. Up to the present stage of development, since a general interpolation scheme for variables belonging to an arbitrary manifold it is not straightforward and it has not been implemented yet, only Euclidean, n-dimensional, parameters can have a limited bandwidth time dependency. Furthermore, such a scheme would have to be implemented relying only on operators defined in Section 3.4.1. However, although we could easily imagine an application in which manifold limited bandwidth state variables were valuable, we have never faced a case in which their lack was a serious issue.

3.5. Functional Description

What follows is a high level description of the functional blocks that implement sensor fusion framework, from the point of view of its information flow. Please refer to Figure 3.6; details for each of the blocks will be given in later sections.

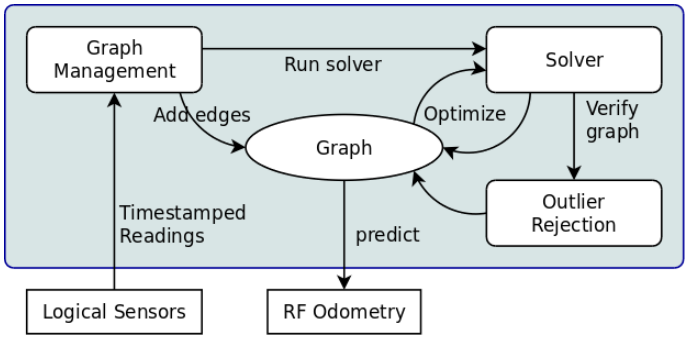


Figure 3.6.: A simple schema of the ROAMFREE processing triggered by the arrival of new measurements from logical sensors.

Data is introduced in the system in the form of timestamped sensor readings; no component is included in ROAMFREE which actually *reads* data from physical sensors. Indeed, the high number of different commercial hardware available, not to mention custom devices developed to handle ad-hoc specifications, would have doomed the sensor acquisition components to cover only a very limited fraction of the use cases. Please note that this task is already undertaken, for instance, by the Robot Operating System (ROS) [69], which is open-source, has a considerable user base and it is becoming a *de-facto* standard in robotics research.

The entry point for timestamped sensor readings consists in the Graph Management component, which is in charge of updating the internal, probabilistic, representation of the information fusion problem, for which we adopted the factor graph formulation: a hypergraph is maintained in which nodes represent robot poses and sensor calibration parameters in a given time window while edges represent sensor measurement constraints (see Section 3.2). As new sensor readings are available, the Graph Management component selects the appropriate sensor model and uses it to build a new factor and inserts it into the graph as it is incident to the proper pose nodes. If needed, e.g., a measurement is newer than the most recent pose in the graph, a new pose node is instantiated for which an initial guess is obtained by means of a Forward Kinematic Logical sensor, if available (see Section 3.3.3). Poses and constraints which are old with respect to the considered time window are discarded and a linearized, equivalent, prior constraint is inserted.

Again by means of the Graph Management component, the user invokes, usually at a fixed rate, a *solver* to be run on the hypergraph, which contains the full description of the sensor fusion problem. The Graph Management thus freezes the graph representation, delaying the

handling of further sensor readings till estimation is completed.

Before the solver is run on the graph, sensor models and other heuristics are employed to perform an *Outlier Rejection* procedure, which is crucial to handle situations in which there exist unmodeled error sources compromising sensor readings. Consider for instance the case in which a wheeled robot performs an intense acceleration: wheel slippage is likely to occur. In this situation, the velocity estimate obtained applying forward kinematics to the wheels encoder readings will probably be inconsistent with other, unaffected, information sources such as a visual odometry system or an accelerometer. Robot motion models usually assume that the forward kinematic yields the vehicle movement with respect to the environment, which is not the case if the robot is slipping. Note that the problem lies in the fact that the error source, i.e., the slippage, is unmodeled. Indeed fusing information sources in a Bayesian way, without eliminating outliers, would yield inaccurate results in which nor the encoders, nor the other information sources are fully trusted. The reconstructed trajectory would lie in the middle between the true one, measured by the inertial sensors, and an inconsistent one based on the wrong assumption of no slippage occurring. In Chapter 5 we will propose a solution, still subject of active research, based on consensus heuristic which aims at selecting a subset of coherent sensor readings in the raw data stream and employ only the selected ones in pose estimation.

Once the solver has completed its tasks on the factor graph, its nodes will contain an estimate of the robot poses in the time window considered, and sensor calibration parameters, based on all the sensor readings available at the time the estimation process was started. Based on this information, the ROAMFREE sensor fusion library accommodates for the estimation latency, in case it is not negligible, predicting the robot pose at the user specified timestamp by means of a Forward Kinematics logical sensor (if available) or extending the reconstructed trajectory assuming constant acceleration. An example of this procedure will be given in Chapter 6.

3.6. Conclusions

In this chapter we have discussed all the key components and techniques employed in the development of the ROAMFREE sensor fusion library, or we have provided references to later chapters for further insights. In the next chapter we will move to the most relevant mathematical topic of this work, which refers to the evaluation of the extended kinematic state at the sensor reference frame as a function of robot poses and the devel-

opment of the logical sensor error models for the multiple measurement domains available in this framework.

Chapter 4

Error Models

As we have discussed in the previous chapter, in ROAMFREE error models are formulated in a hierarchical fashion. At the bottom level we have factor-graph generic edges, which specify the operations required at solver level. Next, *abstract sensors* characterize information sources in terms of placement on the mobile robot and evaluate the kinematic properties of the sensor frame as a function of the robot poses and geometric parameters. [1] Abstract sensors ultimately decouple the representation of the sensor fusion problem, e.g., in terms of robot poses at certain time instants, from the sensor model formulation. Finally, *logical sensor* implement domain specific error models that relate current sensor reading with the kinematic state of the sensor reference frame, evaluated at the abstract sensor level.

In this chapter we discuss the mathematical details behind the core component of the abstract sensor layer, i.e., the *backward augmented state estimator*; it computes all the relevant kinematic properties of the i-th sensor frame, i.e., position, orientation, velocities and acceleration, as a function of three poses of the odometric reference frame O and the fixed transformation from O to the sensor frame S, describing the

[1] In fact, hardware sensors are seldom placed *at* the robot odometric reference frame. Indeed it is quite common that for practical reasons they have to be misplaced and/or misaligned with respect to the robot frame. This has to be taken into account when sensor readings are related to robot poses by means of error functions.

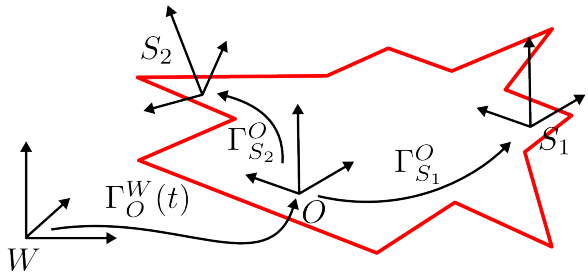


Figure 4.1.: Reference frames employed in the ROAMFREE sensor fusion library. Abstract sensors S_1 and S_2 are misplaced and misaligned with respect to the Odometric reference frame O.

geometric placement of the sensor.

Next, we present each logical sensor available in the framework and how these are defined by an error model building upon the backward augmented state estimator. These error models ultimately relate actual sensor readings with the state representation.

Finally, we discuss how the direction of steepest descent for these error models can be evaluated; this information is often required by most of the non-linear optimization algorithms commonly employed to solve the max-likelihood problem.

4.1. Preliminaries

Let us introduce some details on the notation and on the operators that have been be employed in the development of the abstract and logical sensor models.

4.1.1. Notation

In the following we will refer to three different reference frames, as it has depicted in Figure 4.1. In particular, W is the world fixed reference frame, O is the robot odometric reference frame, and S_i is the i-th sensor reference frame. The goal of the pose tracking problem is to estimate over time the transformation that takes from W to O. In the following we will always refer to one sensor at time and we will omit the i index. It is very important to remark that we model the whole robot as a rigid body. This means that the distance between any two given point on the robot remains constant, thus the transformations from frame O to frame

S_i are given and do not change over time. This assumption allows us to compute the kinematic properties for S_i as a function of the ones for O.

We express rigid transformations Γ_B^A, which moves vectors from B to A, with respect to frame A with its translation part $B^{(A)}$, i.e., the origin of B expressed with respect to A, and its rotational part, R_B^A. Regarding vectors, unless otherwise specified, they are expressed in the reference frame they refer to, e.g., $\omega^{(O)}(t_1)$ is the angular velocity of the O reference frame, with respect to the fixed frame W, at time t_1, expressed in the O reference frame.

Finally, since error functions are evaluated multiple times during GN optimization, when we refer to state variables, e.g. $\Gamma_O^W(t)$, we always intend their current estimate $\breve{\Gamma}_O^W(t)$. Moreover, for the sake of brevity, in cases where it can be deduced by the context, we will leave out the time dependency for the state variables or sensor parameters. Instead, we will employ the following shortcut for referring transformations or other quantities to particular time instants previously mentioned in the context, e.g., if t_1 is a specific time instant, then $\Gamma_O^W(t_1) \equiv \Gamma_{O_1}^W$, in which the subscript is placed at O since W is fixed, or $\omega^{(O)}(t_1) \equiv \omega^{(O_1)}$.

4.1.2. State Variable Operators

As we have anticipated in Section 3.4.1, in ROAMFREE, state variables hide their internal representation under operators which allow both error functions and solvers to operate without having to deal with domain specific issues or take special care to ensure over-parametrization constraints to be satisfied.

In this section discuss each operator in detail. Moreover, along with an informal discussion of the properties of these operators, we develop a reference implementation based on Hamiltonian unit quaternions, which are the default choice for representing 3-DoF rotations in ROAMFREE. In the following, $\mathcal{M}$, represents a generic, non-Euclidean, manifold.

Increment: $\boxplus$

This operator applies an Euclidean increment w to a non-Euclidean variable q, i.e., $\boxplus : \mathcal{M} \times \mathbb{R}^n \to \mathcal{M}$. This operator ensures that after the increment the variable still belongs to the original manifold, i.e., $\mathcal{M}$ is closed with respect to $\boxplus$. If $\mathcal{M} \equiv \mathbb{R}^n$, then $\boxplus \equiv +$.

If $\mathcal{M}$ is a Lie Group, i.e., an algebraic group which is also a differentiable manifold, this operator refers to applying the *exponential map* to w and composing the result with q. Examples of Lie Groups commonly employed in robotics are the space of the 3-DoF rotations SO(3), the spe-

cial Euclidean group SE(3), which models rigid body transformations, and Sim(3), the space of 3-D similarity transformations.

Commutative property does not hold in general for composition on arbitrary manifolds, thus a *left* and a *right* variant of the $\boxplus$ operator can be formulated, depending the order of the aforesaid composition. Indeed, the Euclidean increment w may belong to either the tangent space on the left or on the right with respect to q. To get further insights a discussion of the algebraic structure of the Lie Groups would be required. For such a discussion, please refer to [38] and [73]. For an application of Lie Groups in pose tracking instead see [74].

We propose here a example to clarify the rationale behind the two variants of $\boxplus$ based on the more familiar properties of the rotation matrices. Consider two reference frames, W and O, and the rotation matrix R_O^W encoding the transformation that movements vectors from O to W. Suppose we want to perturb the rotation matrix R_O^W: since it encodes the *relative* rotation between the two frames, if we look from a third reference frame, the same perturbation must be achievable either changing O or W. In the first case we would obtain the $R_{\tilde{O}}^W$ matrix and in the second $R_O^{\tilde{W}}$. More precisely, it holds that:

$$R_{\tilde{O}}^W = R_O^W R_{\tilde{O}}^O = R_W^{\tilde{W}} R_O^W = R_O^{\tilde{W}}. \tag{4.1}$$

where it is possible to see that in the first case a perturbation rotation $R_{\tilde{O}}^O$ post-multiplies our original rotation, it is applied on the right, and it is expressed in the local reference frame O: this fact is related to the right form of the increment operator, $\boxplus_R$. In the second case pre-multiplication is employed, the rotation is composed on the left and the perturbation $R_W^{\tilde{W}}$ is expressed in the global reference frame W, giving $\boxplus_L$. The two perturbations are different, while the resulting rotation is the same. Moreover, there is a relation between the two perturbations: if we post-multiply by $\left(R_O^W\right)^{-1}$ both members in (4.1) we obtain

$$R_W^{\tilde{W}} = R_O^W R_{\tilde{O}}^O \left(R_O^W\right)^{-1}, \tag{4.2}$$

which is related to the so called Adjoint Map and demonstrates the fact that commutative property does not hold for composition in Lie Groups.

In the following, we give the $\boxplus$ formulation employed in ROAMFREE for unit quaternions. Consider $q = [q_w, q_x, q_y, q_z]$ such that $||q|| = 1$. The expression of the perturbed quaternion $\tilde{q} = q \boxplus w$, $w \in \mathbb{R}^3$, can be obtained from the well known differential equation

$$\dot{q} = \frac{1}{2} Q(q)[0, \omega_x, \omega_y, \omega_z]^T + O(|\omega|^2) \tag{4.3}$$

where Q is the matrix representation of the quaternion product operator:

$$Q = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix}. \tag{4.4}$$

Here w can be thought as an angular displacement over the three axis referred with respect to the local frame. Truncating (4.3) to the first order and applying an Euler integration scheme we obtain:

$$q \boxplus_R w: \quad \tilde{q} = q + \frac{1}{2} Q(q)[0, \omega_x, \omega_y, \omega_z]^T. \tag{4.5}$$

This gives the right form of the increment operator, $\boxplus_R$ (we will often omit the R subscript). The left form, $\boxplus_L$, assumes that w lives in the reference frame that is global with respect to q. In case of quaternions, this simply accounts to employ the matrix Q^+, instead of Q in (4.5):

$$Q^+ = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & q_z & -q_y \\ q_y & -q_z & q_w & q_x \\ q_z & q_y & -q_x & q_w \end{bmatrix}. \tag{4.6}$$

$$q \boxplus_L w: \quad \tilde{q} = q + \frac{1}{2} Q^+(q)[0, \omega_x, \omega_y, \omega_z]^T, \tag{4.7}$$

Decrement: $\boxminus$

This operator relates to the Euclidean $-$ and, to some extent, it can be thought as the inverse of the increment operator, i.e., $\boxminus : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^n$ and, given two elements q_1 and q_2 belonging to the manifold $\mathcal{M}$, $\boxminus$ returns the euclidean increment such that:

$$\forall q_2 \in \mathcal{M}: \quad \forall q_1 \boxplus (q_2 \boxminus q_1) = q_2, \tag{4.8}$$

$$\forall w \in \mathbb{R}^n: \quad (q \boxplus w) \boxminus q = w. \tag{4.9}$$

As for the increment operator, two variants can be formulated, depending on whether the result belongs to the tangent space of q_1 or q_2.

Considering unit quaternions, to build the $\boxminus_R$ operator, we observe that a closed form expression for ω can be obtained solving the linear system $Qx = 2(\tilde{q} - q)$ and discarding the first component of x. Similarly, the left form $\boxminus_L$ is obtained replacing Q with Q^+. Note that these expressions hold only if $||\omega||$ is small, i.e., being it a local perturbation. Otherwise, more complex expressions for ω can be employed, e.g., the well known Rodrigues formula.

Default Action on $\mathbb{R}^n$: $(\cdot)$

This operator allows to *apply* an element q belonging to the manifold $\mathcal{M}$ to an Euclidean vector x, i.e., $(\cdot) : \mathcal{M} \times \mathbb{R}^n \to \mathbb{R}^n$. The semantics of this operator clearly depends on the considered manifold. For instance, in case of SO(3), applying an element to an Euclidean vector means rotating the vector. In the writing, when there is no risk of confusion, we will often omit this operator.

In case of unit quaternions, we employ well known formulas to construct a rotation matrix from a quaternion q, which are:

$$R(q) = \begin{bmatrix} q_w^2+q_x^2-q_y^2-q_z^2 & 2(q_xq_y-q_wq_z) & 2(q_wq_y+q_xq_z) \\ 2(q_xq_y+q_wq_z) & q_w^2-q_x^2+q_y^2-q_z^2 & 2(-q_wq_x+q_yq_z) \\ 2(-q_wq_y+q_xq_z) & 2(q_wq_x+q_yq_z) & q_w^2-q_x^2-q_y^2+q_z^2 \end{bmatrix} \tag{4.10}$$

and thus we can define $q \cdot x$ as $x' = R(q)x$, where the usual matrix product takes place.

Inverse: $^{-1}$ **and Composition:** $\circ$

The inverse and the composition operators rely on the hypothesis that the underlying manifold is also a group, and thus the usual properties hold: (i) existence and uniqueness of the identity element for composition, (ii) existence of the inverse for each element of the manifold, (iii) associativity of the composition operator, (iv) closure of the composition operator with respect to the manifold.

The inverse operator is unary, i.e., $^{-1} : \mathcal{M} \to \mathcal{M}$, and returns the element of the manifold such that $q \circ q^{-1} = \mathbf{I}$, where $\mathbf{I}$ is the identity element for the composition operator ($q \circ \mathbf{I} = q$).

In case of unit quaternions, the inverse operator it is defined as

$$q^{-1} = [q_w, -q_x, -q_y, -q_z]. \tag{4.11}$$

Finally, the operator $\circ : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ composes two elements of the manifold, returning another element of the manifold itself, according to the semantics of the variable domain. For unit quaternions, it corresponds to the quaternion product, whose matrix form has been already presented in Equation 4.4, and $q_1 \circ q_2 = Q(q_1)q_2$.

Here we have shown how the non-Euclidean structure of the unit quaternion space can be hidden by a set of operators which define the general interface for state variables. Employing this paradigm in ROAM-FREE, we are able to to decouple internal representations from their manipulation.

4.2. The Backward Augmented State Estimator

Let us consider three robot poses $\Gamma_O^W(t_1)$, $\Gamma_O^W(t_2)$ and $\Gamma_O^W(t_3)$ at different time instants t_1, t_2 and t_3 such that $t_1 < t_2 < t_3$. These poses are a subset of the variables of the estimation problem. Let us also take into account possible displacement and misalignment between frame O and S by means of two calibration parameters $\mathbf{S}^{(O)}$ and $\mathbf{R}_S^O$.

The transformation $\Gamma_O^W = [O^{(W)}, \mathrm{R}_O^W]$ does not explicitly encode information about velocities and accelerations, which can also differ with respect to the ones at S; to account for this, for each sensor and for each pose triplet, we define the *augmented state* as

$$\hat{x}_S(t_3) = \left[S_3^{(W)}, \mathrm{R}_{S_3}^W, S_3^{(S_2)}, \mathrm{R}_{S_3}^{S_2}, v^{(S_3)}, \omega^{(S_3)}, a^{(S_3)}, \alpha^{(S_3)}\right], \tag{4.12}$$

where $\hat{x}_S$ is a function of $\Gamma_{O_1}^W$, $\Gamma_{O_2}^W$, $\Gamma_{O_3}^W$, $\mathbf{S}_i^{(O)}$, $\mathbf{R}_{S_i}^O$ and of $\Delta t_2^1 = t_2 - t_1$ and $\Delta t_3^2 = t_3 - t_2$. More precisely, the components of $\hat{x}_S$ are:

- the position of the sensor at time t_3 with respect to W: $S_3^{(W)}$
- the sensor orientation at time t_3 with respect to W: $\mathrm{R}_{S_3}^W$
- the sensor position at time t_3 with respect to frame S at t_2 in frame S_2, or equivalently, the displacement between S_2 and S_3: $S_3^{(S_2)}$
- the relative rotation between frames S_2 and S_3: $\mathrm{R}_{S_3}^{S_2}$
- the sensor linear velocity at time t_3: $v^{(S_3)}$
- the sensor angular velocity at time t_3: $\omega^{(S_3)}$
- the sensor linear acceleration at time t_3: $a^{(S_3)}$
- the sensor angular acceleration at time t_3: $\alpha^{(S_3)}$

The augmented state encodes all the information needed to relate generic readings in sensor frame S with the variables of the estimation problem. Indeed, $\hat{x}_S$ fully characterizes the kinematic state of the sensor frame S and allows to develop sensor models in a decoupled and hierarchical way. $\hat{x}_S$ answers the following question: *given the current estimates for the robot pose at t_1, t_2 and t_3, and possibly for the calibration parameters $\boldsymbol{S}^{(O)}$ and $\boldsymbol{R}_S^O$, what is the value of the speed, the acceleration, and of the other kinematic quantities, at the sensor frame S at time t_3?* This question applies for each sensor and its answer has nothing to do with the type of information produced by the sensor. This

is why we choose to decouple the modeling of the kinematic state of the sensor and introduce the *backward augmented state estimator*, where the name refers to the fact that a finite difference scheme is employed to estimate quantities at time t_3 based on backward time instants. In the following, we discuss how each component of $\hat{x}_S$ is computed.

Pose of S

The sensor position and orientation at time t_3 are simply computed from $\Gamma^W_{O_3}$, $\mathbf{S}^{(O)}$ and $\mathbf{R}^O_S$:

$$S_3^{(W)} = O_3^{(W)} + \mathrm{R}^W_{O_3}\mathbf{S}^{(O)}, \tag{4.13}$$

$$\mathrm{R}^W_{S_3} = \mathrm{R}^W_{O_3}\mathbf{R}^O_S. \tag{4.14}$$

Displacement between S_2 and S_3

The displacement between sensor frame at time t_2 and t_3, expressed with respect to sensor frame S_2, or equivalently, the position of the origin on S_3 with respect to S_2, is evaluated as:

$$\begin{aligned} S_3^{(S_2)} &= \left(\mathrm{R}^W_{S_2}\right)^{-1}\left(S_3^{(W)} - S_2^{(W)}\right) \\ &= \left(\mathrm{R}^W_{O_2}\mathbf{R}^O_S\right)^{-1}\left(\left(O_3^{(W)} + \mathrm{R}^W_{O_3}\mathbf{S}^{(O)}\right) - \left(O_2^{(W)} + \mathrm{R}^W_{O_2}\mathbf{S}^{(O)}\right)\right), \end{aligned} \tag{4.15}$$

$$\mathrm{R}^{S_2}_{S_3} = \left(\mathrm{R}^W_{S_2}\right)^{-1}\mathrm{R}^W_{S_3} = \left(\mathrm{R}^W_{O_2}\mathbf{R}^O_S\right)^{-1}\left(\mathrm{R}^W_{O_3}\mathbf{R}^O_S\right). \tag{4.16}$$

Velocity of S

Here we derive the estimators for the linear and angular velocities, assuming that for all $t \in [t_2, t_3]$ they are constant with respect to the fixed frame W. This assumption deserves to be discussed in detail, since in many situations there exist a correlation between orientation and linear velocity. In other words, if the vehicle orientation with respect to W changes, then also the direction of the linear velocity changes by the same amount. In these situations the linear velocity can be considered constant with respect to the moving frame O, and not to W, at least in a local neighborhood of t_3.

Consider for instance the motion of a non-holonomic vehicle, such a differential drive wheeled robot: if no slippage occurs, when the robot turns, also the direction of its linear velocity changes due to the interaction of the wheels with the floor. More accurate estimators for the linear velocity could be formulated if this fact was taken into account.

Unfortunately, this is not true for holonomic vehicles, such as omnidirectional or flying robots, thus, for the sake of generality, and to avoid to introduce platform dependent assumptions, we stay with the more general case in which no correlation is assumed between orientation and linear velocity. Furthermore, note that this subtlety can be neglected once the sampling rate is sufficiently high.

We model the pose of the robot between times t_2 and t_3 with the following continuous time differential equations:

$$\begin{cases} \dot{O}^{(W)}(t) = v^{(W)} & (4.17a) \\ \dot{R}_O^W(t) = \omega^{(W)}, & (4.17b) \end{cases}$$

where the velocities $v^{(W)}$ and $\omega^{(W)}$ are constant in the time interval considered, when referred to the W frame. The solution for the initial value Cauchy problem is given by

$$\begin{cases} O^{(W)}(t) = O^{(W)}(t_0) + v^{(W)}(t - t_0) & (4.18a) \\ \mathrm{R}_O^W(t) = \mathrm{R}_O^W(t_0) \boxplus_L \left(\omega^{(W)}(t - t_0)\right), & (4.18b) \end{cases}$$

in which, since the angular velocity is expressed with respect to the fixed frame, the left form of the *boxplus* operator appear, see Section 4.1.2. Given a couple of successive poses, $\Gamma_{O_2}^W$ and $\Gamma_{O_3}^W = [O_3^{(W)}, \mathrm{R}_{O_3}^W]$, we can solve (4.17a) and (4.17b) in $v^{(W)}$ and $\omega^{(W)}$:

$$\begin{cases} v^{(W)} = \dfrac{1}{\Delta t_3^2}\left(O_3^{(W)} - O_2^{(W)}\right) & (4.19a) \\ \omega^{(W)} = \dfrac{1}{\Delta t_3^2}\mathrm{R}_{O_3}^W \boxminus_L \mathrm{R}_{O_2}^W. & (4.19b) \end{cases}$$

The computed quantities $v^{(W)}$ and $\omega^{(W)}$ refer to O but are expressed with respect to W. The final step consists in computing $v^{(S)}$ and $\omega^{(S)}$, i.e., the linear and angular velocities of the sensor frame with respect to W, expressed with respect to S. To this end, we employ the rigid body assumption:

$$v^{(S_3)} = \left(\mathbf{R}_S^O\right)^{-1}\left(\mathrm{R}_{O_3}^{W\,-1} v^{(W)} + \left(\mathrm{R}_{O_3}^{W\,-1}\omega^{(W)}\right) \times \mathbf{S}^{(O)}\right), \tag{4.20}$$

$$\omega^{(S_3)} = \left(\mathrm{R}_{O_3}^W \mathbf{R}_S^O\right)^{-1}\omega^{(W)}. \tag{4.21}$$

Acceleration of S

In this section we derive estimators for $a^{(S)}$ and $\alpha^{(S)}$. To this end, we can not assume that the velocities are constant, as we did in the previous

section, otherwise no acceleration would occur. Instead, we assume that the accelerations are constant with respect to W for each $t \in [t_1, t_3]$.

To derive the required estimators, we extend the differential equations in (4.17a) and (4.17b) as:

$$\begin{cases} \dot{v}^{(W)}(t) = a^{(W)} & (4.22a) \\ \dot{\omega}^{(W)}(t) = \alpha^{(W)} & (4.22b) \\ \dot{O}^{(W)}(t) = v^{(W)}(t) & (4.22c) \\ \dot{R}_O^W(t) = \omega^{(W)}(t), & (4.22d) \end{cases}$$

where this time both $v^{(W)}$ and $\omega^{(W)}$ change over time, then we move to discrete time by means of the Runge-Kutta R2 numerical integration scheme: for differential equations of the form $\dot{y}(t) = f(y, t)$, it holds that

$$\begin{aligned} k_1 &= \Delta t f(t_n, y_n), \\ k_2 &= \Delta t f(t_n + \frac{\Delta t}{2}, y_n + \frac{1}{2}k_1), \\ y_{n+1} &= y_n + k_2 + \mathcal{O}(\Delta t^3). \end{aligned} \tag{4.23}$$

In our case this gives:

$$\begin{cases} v^{(W_{n+1})}(t) = v_n^{(W)} + a^{(W)}\Delta t_{n+1}^n & (4.24a) \\ \omega^{(W_{n+1})}(t) = \omega_n^{(W)} + \alpha^{(W)}\Delta t_{n+1}^n & (4.24b) \\ O_{n+1}^{(W)} = O_n^{(W)} + \left(v_n^{(W)} + a^{(W)}\frac{\Delta t_{n+1}^n}{2}\right)\Delta t_{n+1}^n & (4.24c) \\ \mathrm{R}_{O_{n+1}}^W = \mathrm{R}_{O_n}^W \boxplus_L \left[\left(\omega_n^{(W)} + \alpha^{(W)}\frac{\Delta t_{n+1}^n}{2}\right)\Delta t_{n+1}^n\right]. & (4.24d) \end{cases}$$

Writing Equation 4.24c for $n = 2$ and $n = 3$, together with Equation 4.24b for $n = 2$, and solving with respect to $\alpha^{(W)}$ we obtain an estimator for the linear acceleration of frame O with respect to W:

$$a^{(W)} = \left(\frac{2}{\Delta t_2^1 + \Delta t_3^2}\right)\left(\frac{O_3^{(W)} - O_2^{(W)}}{\Delta t_3^2} - \frac{O_2^{(W)} - O_1^{(W)}}{\Delta t_2^1}\right), \tag{4.25}$$

and, similarly, we obtain also the estimator for the angular acceleration:

$$\alpha^{(W)} = \left(\frac{2}{\Delta t_2^1 + \Delta t_3^2}\right)\left(\frac{\mathrm{R}_{O_3}^W \boxminus_L \mathrm{R}_{O_2}^W}{\Delta t_3^2} - \frac{\mathrm{R}_{O_2}^W \boxminus_L \mathrm{R}_{O_1}^W}{\Delta t_2^1}\right). \tag{4.26}$$

As it happened for linear and angular velocities, from $a^{(W)}$ and $\alpha^{(W)}$ we compute the accelerations for the sensor frame S employing the rigid

body assumption:

$$a^{(S_3)} = \left(\mathbf{R}_S^O\right)^{-1} \Big(\mathrm{R}_{O_3}^{W\,-1} a^{(W)} + \omega_3^{(W)} \times \omega_3^{(W)} \times \mathbf{S}^{(O)} + \\ + \left(\mathrm{R}_{O_3}^{W\,-1} \alpha^{(W)}\right) \times \mathbf{S}^{(O)} \Big), \tag{4.27}$$

$$\alpha^{(S_3)} = \left(\mathrm{R}_{O_3}^W \mathbf{R}_S^O\right)^{-1} \alpha^{(W)}. \tag{4.28}$$

where $\omega_3^{(W)}$ is computed from $\omega_2^{(W)}$ and $\alpha^{(W)}$ by means of (4.24b).

4.3. Logical Sensors

As we have anticipated in Section 3.3, raw sensor data is handled by coupling each hardware sensor, or software algorithm, producing the measurements with a logical sensor that models the corresponding domain and offers parameters to account for sources of bias, distortion or other sensor specific properties.

For convenience, we recall Equation 3.6, which gives the general form of the logical sensor error models:

$$e(t) = \hat{z}\left(t; \hat{x}_S(t), \xi\right) \boxminus \left(z(t) \boxplus \eta\right), \tag{4.29}$$

where $\hat{z}(\cdot)$ is a measurement predictor computed as a function of the augmented state for sensor frame S, $\hat{x}_S$, and of further, measurement domain dependent, calibration parameters ξ. In the same equation, z is the current sensor reading at time t, and η is a zero-mean Gaussian noise with covariance Σ_η. Here the $\boxplus$ and $\boxminus$ operators are employed because the measurement domain might not be Euclidean: for instance, for an inertial measurement unit that ships an on-board attitude tracking algorithm, $z \in \mathrm{SO}(3)$.

In the following we develop the measurement predictors $\hat{z}$ for all the logical sensors currently available in ROAMFREE. As we have shown in Equation 4.29, these predictors are functions, in certain cases immediate, of the components of the backward augmented state estimator and they compute a prediction for the sensor readings based on up to three robot poses and a sensor dependent set of calibration parameters.

4.3.1. Absolute Position

This logical sensor allows the handling of absolute position information sources with respect to a world fixed reference frame, such as a GPS sensor or a SLAM system:

$$\hat{z} = S^{(W)}. \tag{4.30}$$

Note that the readings coming from a GPS sensor, in the form of Latitude, Longitude and Altitude, can not be directly employed in Equation 4.30. Typically, a plane tangent to the WGS84 reference ellipsoid at a given reference point it is chosen and the GPS readings are projected onto it, resulting in an East-North-Up coordinate system which can be considered Euclidean in a neighborhood of the linearization point.

4.3.2. Absolute Orientation

Absolute orientation readings with respect to a fixed frame, such as the ones coming from attitude tracking systems available on most of the commercial Inertial Measurement Units, are handled by means of this logical sensor. A predictor for the sensor readigs reads as:

$$\hat{z} = \mathrm{R}_S^W. \tag{4.31}$$

Note that in this case $\hat{z} \in \mathrm{SO}(3)$.

4.3.3. Linear Velocity

This logical sensor handles sources of linear velocity readings, which range from wheel encoders to scan-matching and visual odometry:

$$\hat{z} = \mathbf{k} v^{(S)}, \tag{4.32}$$

where $\mathbf{k}$ is a scalar. In case this logical sensors is employed to handle linear velocity estimates coming from a monocular visual odometry system, the $\mathbf{k}$ parameter is enabled for online tracking and it accounts for the fact that estimates are not expressed in a metric scale. Note that in this case this sensor constraints only the *direction* of the linear velocity, while its magnitude has to be estimated from other sources.

4.3.4. Angular Velocity

This logical sensor handles sources of angular velocity readings such as gyroscope sensors and visual odometry systems. Gyroscopes are known to be affected by a time-varying bias and a scale factor, which are handled by means of two calibration parameters.

$$\hat{z} = \mathbf{A}\omega^{(S)} + \mathbf{b}, \tag{4.33}$$

where A is a diagonal matrix and $\mathbf{b}$ is a bias vector.

4.3.5. Linear Acceleration

As for the angular velocity, also the linear acceleration sensor model takes into account gain and bias calibration parameters. Moreover, the Earth gravitational acceleration vector $\vec{g}$, which is assumed constant and known in frame W, has to be considered.

$$\hat{z} = \mathbf{A}\left(a^{(S)} + \left(\mathrm{R}_S^W\right)^{-1}\vec{g}^{\,(W)}\right) + \mathbf{b}, \tag{4.34}$$

where $\mathbf{A}$ is a diagonal matrix and $\mathbf{b}$ is a bias vector.

4.3.6. Vector Field

This logical sensor allows to handle sensors which measure a vector field existing in the operation area. One remarkable example is the Earth magnetic field $\vec{h}$. As for $\vec{g}$, we assume that the vector field is uniform and constant over the whole operation area.

Here we employ an error model which targets magnetometer sensors, being those the most common sources for this logical sensor. It is known that the readings from this kind of sensors are substantially affected by the ferromagnetic properties of the materials in their surroundings. Two major effects apply here: (i) hard iron bias coming from the combined result of the permanent magnets inherent to the robot structure, as well as other elements installed in the robot which is constant in the S reference frame; (ii) soft iron effects, resulting from the interaction of an external magnetic field with the ferromagnetic materials in the vicinity of the sensor. The resulting magnetic field in the latter case depends on the magnitude and direction of the applied magnetic field with respect to the soft iron material.

Here we employ an error model which keeps into account hard and soft iron distortion, non-orthogonality of the sensor axes, scaling, bias and misalignment with respect to O [83]. The predicted magnetometer measurement reads as:

$$\hat{z} = \mathbf{A}\left(\mathrm{R}_S^W\right)^{-1}\vec{h}^{\,(W)} + \mathbf{b}. \tag{4.35}$$

where $\mathbf{A}$ is an unconstrained 3×3 matrix and $\mathbf{b}$ is a bias vector. These parameters can be determined by the frameworks employing long time lags and trajectories able to fully excite all the available degrees of freedom. We will discuss an example of such calibration in Section 6.2.

4.3.7. Landmark Position and/or Orientation

This sensor allows to handle the case in which a sensor estimates the relative position and/or orientation of a world fixed landmark with respect

to the sensor itself. An example is given by fiducial marker tracking algorithms, such as ARTag [31], which we will discuss in Chapter 7.

This kind of sensors measures the transformation between the sensor frame and the landmark frame L, for which one or both component may be considered:

$$\hat{z} = \begin{bmatrix} \hat{L}^{(S)} \\ \hat{R}_L^S \end{bmatrix} = \begin{bmatrix} \left(\mathrm{R}_S^W\right)^{-1} \mathbf{L}^{(W)} \\ \left(\mathrm{R}_S^W\right)^{-1} \mathbf{R}_L^W \end{bmatrix}. \tag{4.36}$$

Note that the transformation between W and L, i.e., the absolute pose of the landmarks with respect to the world frame, is a calibration parameter for this logical sensor; it is seldom known a priori and it is often left to be estimated from sensor readings, as it happens in SLAM algorithms.

4.4. Kinematic Models

In this section we discuss the kinematic models implemented in ROAMFREE. As discussed in Section 3.3.3 these sensors allow for predictors $\hat{\Gamma}_O^W(t+1)$ to be formulated as a function of $\Gamma_O^W(t)$ and $z(t)$.

Since it is is common practice to put the O reference frame at the point on the robot whose motion is modeled by kinematics equations, in the following we will assume that $O \equiv S$. In other words, for kinematic logical sensors, we will assume that $\mathbf{S}^{(O)} = \mathbf{0}$ and $\mathbf{R}_S^O = \mathbf{I}$.

At the present stage of development, the kinematic models available in the framework focus on wheeled robots. These models assume that the motion of the robot is 3-DoF, thus predictions will be available only for the x and y components of linear velocities and accelerations, and for the z component of the angular velocity. Since ROAMFREE tracks 6-DoF motion, we augment each measurement vector with the following dummy readings:

$$z = \begin{bmatrix} v_z^{(S)} \\ \omega_x^{(S)} \\ \omega_y^{(S)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.37}$$

These readings state that the robot motion is 3-DoF in a local neighborhood of its 6-DoF trajectory. The strength of this constraints can be tuned by means of the noise vector covariance associated to these components of the measurement vector, so that the kinematic models can be employed also if the assumption that the motion is 3-DoF does not hold on a global scale.

4.4.1. Differential Drive

In differential drive kinematics, encoders read the left and the right wheel speed, $z = [\omega_l, \omega_r]^T$. In order to write an error model in the form of (4.29), we need to compute a prediction $\hat{z}$ for the wheel speed based on the robot linear and angular velocities. This prediction read as:

$$\hat{z} = \begin{bmatrix} v_y^{(S)} \\ \omega_r \\ \omega_l \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{\mathbf{r}}\left(v_x^{(S)} + \frac{\mathbf{B}\omega_z^{(S)}}{2}\right) \\ \frac{1}{\mathbf{r}}\left(v_x^{(S)} - \frac{\mathbf{B}\omega_z^{(S)}}{2}\right) \end{bmatrix}. \tag{4.38}$$

where $\mathbf{r}$ and $\mathbf{B}$ are two calibration parameters, respectively the wheel radius and the robot baseline. Note that differential drive robots are nonholonomic, i.e., they have fewer degrees of freedom with respect to their environment. In fact, in absence of slippage, they cannot translate along the y direction. This motivates the extra dummy component in Equation 4.38, which is similar to the ones in (4.37).

4.4.2. Ackermann Steering Geometry

Four wheel vehicles usually employ Ackermann steering geometry to solve the problem of wheels on the inside and outside of a turn needing to trace circles of different radii. The S reference frame, which we assume to coincide with O, is placed at the middle point of the rear wheels axis.

Here we employ a first order approximation of the dynamic equations which govern the motion of these vehicles which assume infinite cornering stiffness, i.e., no lateral slippage, and model the four wheel vehicles as an equivalent *bycicle* with two wheels, for details see [1]. Moreover, encoders are usually placed such that they read the handlebar or the steering wheel angle, which often does not coincide with the actual turning angle of the vehicle, due to reductions and geometry of the steering mechanisms. Here we assume that the non-linear function that maps steering wheel angles to turning angles of the equivalent vehicle model can be replaced with its first order expansion.

For this logical sensor, the z vector consists in the vehicle tangential speed v_a, which is usually estimated from rear wheel encoders, and the steering wheel position δ, plus a constraint on the ortogonal speed $v_y^{(S)}$ which encodes the fact that the vehicle is nonholonomic and that infinite cornering stiffness is assumed:

$$\hat{z} = \begin{bmatrix} v_y^{(S)} \\ v_a \\ \delta \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{k}_v v_x^{(S)} \\ \frac{1}{\mathbf{a}_s}\left[\arctan\left(\frac{\mathbf{L}\omega_z^{(S)}}{\mathbf{k}_v v_x^{(S)}}\right) - \mathbf{b}_s\right] \end{bmatrix}. \tag{4.39}$$

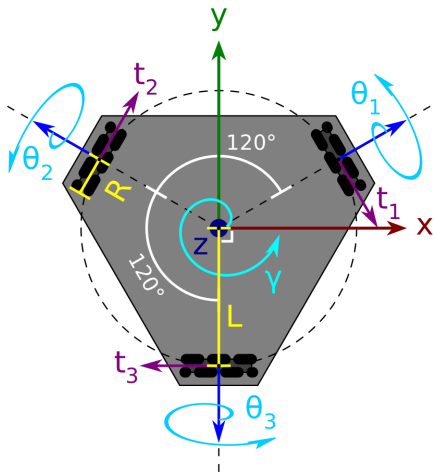


Figure 4.2.: The three-wheels omnidirectional kinematic.

Here $\mathbf{a}_s$ and $\mathbf{b}_s$ are two parameters which characterize the linearization of the steering angle map, i.e., the turning angle δ' is related with the handlebar angle δ by the linear relation $\delta' = \mathbf{a}_s\delta + \mathbf{b}_s$. Moreover, $\mathbf{k}_v$ is a gain factor that can be enabled, depending on how the tangential speed is measured, and eventually set to be estimated, for instance in case reductions are not known. Finally, $\mathbf{L}$, is the distance between vehicle front and rear wheel axes.

As we have seen at the beginning of the section, the dummy measurement $v_y^{(S)} = 0$ holds locally: considering that every Ackermann kinematic reading is augmented with this component, its meaning should be understood as: *the vehicle, on average, does not move sideways.* Since a covariance matrix is associated to sensor readings, it is possible to tune how strong this constraint should be.

4.4.3. Omnidirectional

Here we consider a three-wheeled omnidirectional kinematic model. In these platforms, wheels are of omni type, with several rollers on their perimeter, to give traction on the component orthogonal to their axis while allowing free movements in the axis direction; this enabling movements with three degrees of freedom (Please see Figure 4.2).

Encoders are usually employed to read the wheel angular velocity. We employ the inverse kinematic to formulate a predictor of these quantities

as a function of the robot linear and angular velocity:

$$\hat{z} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \frac{1}{\mathbf{R}} \begin{bmatrix} \cos\left(\frac{\pi}{3}\right) & -\cos\left(\frac{\pi}{6}\right) & -1 \\ \cos\left(\frac{\pi}{3}\right) & +\cos\left(\frac{\pi}{6}\right) & -1 \\ -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_x^{(S)} \\ v_y^{(S)} \\ \mathbf{L}\omega_z^{(S)} \end{bmatrix}, \tag{4.40}$$

where $\mathbf{R}$ is the wheel radius and $\mathbf{L}$ is the distance between each wheel and the odometric center of the robot, which is placed at the intersection of the wheel axes.

4.5. Jacobians

As we have discussed in Section 3.3, the root of the sensor model hierarchy consists in generic factor-graph edges that expose the two operations solvers need to perform state estimation: the first one evaluates errors as a function of the backward augmented state estimator and the current sensor reading, according to the logical sensor models in Section 4.3. The second one instead provides the solver with a notion of the error function direction of steepest descent, in terms of Jacobian matrices, which are required by many non-linear optimization algorithms such as Gauss-Newton and Levenberg-Marquardt, currently employed in ROAMFREE. It remains to discuss how Jacobians are evaluated and how we take advantage of the decoupling between state variables representation, abstract sensors, and logical sensors. The characterization of the uncertainty of error models, in terms of a noise covariance matrix, has also to be provided in order to evaluate the likelihood functions associated to each factor; this is done linearly propagating the measurement noise onto the logical sensors error functions.

4.5.1. Error Jacobians with Respect to the Noise

Here we discuss how the edge information matrices are computed, which are needed to associate a likelihood function to each edge, and ultimately to take into account measurement uncertainty in estimation, as we have discussed in Section 3.2.2.

Suppose for now that $z \in \mathbb{R}^n$; Equation 4.29 reduces to:

$$e(t) = \hat{z}(t) - z(t) + \eta, \tag{4.41}$$

where for simplicity we have omitted the dependencies on state variables. It is possible to see that in this case the information matrix $\Omega = \Sigma_\eta^{-1}$ and that the expected value of the error function $\mathbb{E}[e] = e(t)|_{\eta=0}$.

In the general case the measurement domain is a generic manifold $\mathcal{M}$, so we come back to (4.41) by means of the $\boxplus$ operator and propagating Σ_η on $e(t)$ through linearization. More precisely, we replace the general form of the error functions (recall Equation 4.29) with:

$$e(t) = \hat{z}(t) \boxminus z(t) + \eta', \tag{4.42}$$

where an additive noise vector η' replaces the original one such that:

$$\mathbb{E}[e(t)] = \mathbb{E}[\eta'] = 0, \tag{4.43}$$

$$\Sigma_{e(t)} = \Sigma_{\eta'} = J_e(\eta)|_{\eta=0} \, \Sigma_\eta \, J_e(\eta)|_{\eta=0}^T . \tag{4.44}$$

Here $J_e(\eta)$ is the Jacobian matrix of the error function with respect to the noise vector η. Note that the measurement predictor $\hat{z}$, which depends on the backward augmented state estimator, and in turn on the state variables, is treated as a constant when differentiating in η, thus $J_e(\eta)$ can be computed at logical sensor level, and only the value of $\hat{z}$ and z are needed for its evaluation.

Equations 4.43 and 4.44 do not hold in general since the measurement noise affects $e(t)$ through the operators $\boxplus$ and $\boxminus$, whose definition is arbitrary and possibly non-linear. However, these expressions can be justified assuming that $\hat{z}$ is unbiased and observing that, in practice, if η is zero-mean then $\mathbb{E}[x \boxplus \eta] = x$, $x \in \mathcal{M}$.

4.5.2. Error Jacobians with Respect to State Variables

Now consider one instance of an error function $e(\cdot)$ as in Equation 4.42 at a certain time t. As we have seen, error functions are formulated relying on $\hat{z}$, which yields the expected sensor reading given the current estimate of state variables such as robot poses and sensor calibration parameters. This dependency is indirect: first robot poses and geometric placement parameters are employed to compute the backward augmented state estimator $\hat{x}_S$, then the measurement predictor is evaluated as a function of its components. Once we have made explicit these dependencies we obtain:

$$e = \hat{z}\left(\Gamma^W_{O_1}, \Gamma^W_{O_2}, \Gamma^W_{O_3}, \mathbf{S}^{(O)}, \mathbf{R}^O_S, \xi\right) \boxminus z + \eta', \tag{4.45}$$

where we have omitted time dependency since the error function does not depend on the absolute timestamps of the sensor readings. The transformations $\Gamma^W_{O_k}$, with $k \in [1, 2, 3]$, are three robot poses picked according to the sensor reading timestamp. This error function depends on $\Delta t^1_2 = t_2 - t_1$ and $\Delta t^2_3 = t_3 - t_2$, as we have seen in Section 4.2, which are known and constant given the selected robot poses.

From the definition of the $\boxminus$ operator, it follows that $e(\cdot)$ is an Euclidean vector, $e \in \mathbb{R}^J$, and J depends on the specific sensor model. Suppose for a moment that we could stack together all the singleton state variables x_i, with $i \in [1, ..., N]$ appearing in $e(\cdot)$. In that case, the required direction of steepest descent, i.e., the Jacobian matrix of $e(\cdot)$ with respect to x_i, would be given by:

$$J_e(x_1, ..., x_N) = \begin{bmatrix} \frac{\partial e_1}{\partial x_1} & \cdots & \frac{\partial e_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_J}{\partial x_1} & \cdots & \frac{\partial e_J}{\partial x_N} \end{bmatrix} \tag{4.46}$$

To evaluate $\partial e_j / \partial x_i$ one needs to known the full dependency of e_j with respect to the singleton component x_i. However, in ROAMFREE, error functions are formulated in a layered fashion and each e_j depends on the components of the backward state estimator $\hat{x}_S$, which in turn are computed as a function of the actual state variables of the estimation problem. Moreover, x_i is the i-th component of the internal state representation of some multidimensional variable belonging to an arbitrary manifold. As we have discussed in Section 3.2, solvers access to state variables through the operators interface, thus the derivative of the error function with respect to each singleton component of the state variable internal representation is not what we are looking for. Instead we need a notion of the direction of steepest descent that is compatible with the increment operator $\boxplus$, which is the one employed to update variable values during estimation.

We first decouple the logical sensor and the abstract sensor Jacobians by the well known chain rule: being x the set of state variables appearing in $e(\cdot)$, if we rearrange Equation 4.45, replacing the $\boxminus$ operator and the error model with a two argument function $f(\cdot, \cdot)$, and collapsing the backward augmented state estimator in $g(x)$, we obtain:

$$e = f(\hat{x}_S, z) + \eta' = f(g(x), z) + \eta', \tag{4.47}$$

Then, the derivative of the j-th component of e with respect to x_i is given by

$$\frac{\partial e_j}{\partial x_i} = \sum_{k=1,...,K} \frac{\partial f_h}{\hat{x}_{S,k}}\bigg|_{\hat{x}_{S,k}=g_k(x)} \cdot \frac{\partial \hat{x}_{S,k}}{\partial x_i}, \tag{4.48}$$

where $\hat{x}_{S,k}$ is the k-th component of the augmented state.

In our case we have the logical sensor error function f, which depends on the components $\hat{x}_{S,k}$ of the backward augmented state estimator, which are: $S_3^{(W)}$, $\mathrm{R}_{S_3}^{W}$, $S_3^{(S_2)}$, $\mathrm{R}_{S_3}^{S_2}$, $v^{(S_3)}$, $\omega^{(S_3)}$, $a^{(S_3)}$, $\alpha^{(S_3)}$. In turn,

each of these component depend on the state variables x_i: $\Gamma^W_{O_1}$, $\Gamma^W_{O_2}$, $\Gamma^W_{O_3}$, $\mathbf{S}^{(O)}$, $\mathbf{R}^O_S$. Note that $\hat{x}_{S,k}$ and x_i are in general multi-dimensional and non-Euclidean, and that in general $\hat{z}$ depends on few components of $\hat{x}_S$, for instance the absolute position logical sensor relies only on $S^{(W)}_3$ and $\mathrm{R}^W_{S_3}$. Moreover, these components of the augmented state can be computed relying only on $\Gamma^W_{O_3}$ and $\mathbf{S}^{(O)}$ (see Section 4.3.1). This means that $\partial f/\partial \hat{x}_{S,k}$ is zero for multiple k, depending on the particular error function, and the same holds for $\partial \hat{x}_{S,k}/\partial x_i$. However, these sparsity patterns are known for each logical sensor and the framework is able to evaluate only the non-zero component of the summation in (4.48).

In this way the derivatives of the logical sensor error function can be evaluated once the current value of the backward augmented state estimator is known, without having to known its actual expression or how it is computed. However, still x_i belongs to an arbitrary manifold in the general case and thus $\partial \hat{x}_{S,k}/\partial x_i$ has to be evaluated properly .

Here we follow the approach presented in [54] and in [43]:, given that $\hat{x}_{S,k} = g_k(x_1, ..., x_i, ..., x_N)$, i.e, $\hat{x}_{S,k}$ is a certain function of the state variables x_i, we can rephrase the usual derivative definition by means of the difference quotient to rely on the $\boxplus$ and $\boxminus$ operators:

$$\frac{\partial \hat{x}_{S,k}}{\partial x_i} = \lim_{\Delta x \to 0} \frac{g_k(x_1, ..., x_i \boxplus \Delta x, ..., x_N) \boxminus g_k(x_1, ..., x_N)}{\Delta x}. \tag{4.49}$$

which informally can be thought as the local variation in $\hat{x}_{S,k}$ when the non-Euclidean variable x_i is perturbed according to the semantic of its domain by an Euclidean increment of Δx. However, this expression is commonly replaced with:

$$\frac{\partial \hat{x}_{S,k}}{\partial x_i} = \left. \frac{\partial g_k(x_1, ..., x_i \boxplus \Delta x, ..., x_N)}{\partial \Delta x} \right|_{\Delta x = 0}, \tag{4.50}$$

A number of subtleties arise in the approximation in (4.50), in particular we have employed the fact that the limit of the difference quotient, when Δx approaches to zero, can be obtained evaluating the conventional derivative in $\Delta x = 0$. This is not true in general, however this approximation has been employed in other works and implementations, such as the ones referenced before, and it has been shown to yield reasonable results in practice.

It remains to discuss how $\partial f/\partial \xi_h$ are evaluated, where ξ_h is one component of the sensor specific parameter vector ξ. As it happened in Section 4.5.1, when computing the partial derivative with respect to ξ_h, the components of the augmented state y_j are treated as a constant, so we only need their value. Moreover, since ξ_h in general belongs to an arbitrary manifold, we have to employ the scheme in (4.50).

As it has been discussed in Chapter 3, the framework can be instantiated with different state variable representations. This specifies, along with the state variable operator interface, also the state variable traits such as increment size and size of the internal variable representation. The analytic expression of the error functions and of each component of the Jacobian matrices are derived once the state variable representation and the implementation for their operators are known. To optimize the evaluation of such expression, which are sometimes rather complex, we manipulate analytic expressions to perform Common Subexpression Elimination [23].

Chapter 5

Outlier Rejection

To enhance pose tracking accuracy and robustness, multiple sensors are often employed and redundant information is handled by means of sensor fusion algorithms, such as Bayesian filters or, as we do in this work, by means of fixed-lag smoothers based on factor graphs. Unfortunately, sensor readings often include *outliers*, i.e., readings whose likelihood is low given the true robot motion, the observation model and its uncertainty characterization. Measurement outliers are ultimately related to effects that are not taken into account in the sensor models but, being employed to infer the state of the system, which is unknown, their effect is, in general, to introduce biases in the state estimate.

Let us consider a magnetometer sensor: we known that its readings are distorted because of ferromagnetic properties of surrounding objects. If we characterize this distortion and introduce it into the model, the correct sensor orientation can be estimated properly from distorted magnetic field readings. Conversely, if we employ a simpler model, distorted readings will appear to be generated from a sensor in a different, incorrect, orientation.

From this example it becomes clear how outliers are the result of unmodeled phenomena. Nevertheless, more complex models will not solve the outlier issue in general case. Typically, reasonably simple ones are employed so that they relate the sensor readings with the robot state. The goal of outlier rejection is to detect when these relations do

not hold.

Let us indulge in another example to clarify this point. Consider a forward kinematic model for a differential drive robot. If no slippage is assumed, its formulation is straightforward and it relates wheel speeds, read by encoders, to the robot velocity. However, when slippage occurs the encoders will produce outlier readings that, according to the model, lead the estimation algorithm to infer a robot movement, which actually does not exist. A more complex model that takes into account slippage might be able to produce consistent velocity estimates also in case of slippage but requires the current value of the friction coefficient. Concerns arise about how to estimate this parameter.

To handle the presence of outliers, where it is possible, it is common to have redundant sensors of the same type since the probability that an unforeseen event affects all the readings reduces as the number of redundant sensors increases. Indeed, when multiple, homogeneous, information sources are available, statistic consensus heuristics can be employed to detect outliers. One example of this technique is Receiver Autonomous Integrity Monitoring (RAIM, see [44]), for GPS/Galielo localization systems, where the redundancy is available as more than four satellites are visible, allowing to detect multi-path effects and other unmodeled phenomena.

However, the GPS case, in which a moderately high number of homogeneous information sources is available, is quite uncommon. In a general sensor-fusion scenario, sensor readings belong to different domains and are not comparable in a trivial way. Moreover, sensor are misplaced and misaligned with respect to the odometric center of the robot, thus, even homogeneous sensors produce different readings. Consider, for instance, two accelerometers mounted at different places on the mobile robot: the normal component of the acceleration with respect to the robot trajectory depends on the sensor placement and has to be taken into account to compare the readings of the two sensors. Another issue lies in the fact that, although a certain degree of information redundancy is always available, sensors work at different rates and they are seldom synchronized, so, if a reading for a given sensor is available at time t, it is very unlikely that an homogeneous, or comparable, reading from another sensor is also available and sufficiently near to t.

Another technique commonly employed, e.g., in [67], or, in SLAM data association [57, 68], is to check the consistency of the new sensor readings with respect to the current state by means of statistical tests, such as Malhanobis distance or Normalized Innovation Squared (NIS), and employ measurements in the state estimation process only if these test are passed. However, examples can be constructed in which, if an

outlier fails to be detected by chance, it is ultimately able to bias the estimation to such an extent that future inlier readings are considered as outliers with respect to the resulting state belief.

Least-squares estimators as the ones employed in this work, are known to be very sensitive to outliers [46]. Robustified alternatives have thus been proposed, such as M-estimators and the *penalized trimmed squares estimator* [85], which relies on a NP-hard, mixed-integer programming problem, aiming at minimizing the squared-residuals plus a penalty for discarding measurements. In the graph-SLAM community, where the outlier problem substantially reduces to false loop closure constraints inserted by the front end, two popular alternatives, among the others, are *switchable constraints* [76], where latent variables are responsible of enabling or disabling constraints, and *dynamic covariance scaling* [2], were residuals are re-weighted so that a gradient always exists towards their minimization and it gradually increases in the presence of more mutually consistent constraints. More recently, the idea that the outlier rejection problem can be formulated as the search for a maximal cardinality subset of the available measurements that are internally "coherent" appeared in the literature, as in the work by Carlone et Al. [15].

In our work we propose an outlier rejection mechanism, still subject of active research, inspired by Random Sample Consensus (RANSAC, [32]). Differently with respect to current literature, we do not check for consistency with respect to the current state estimate, instead we fit a low-dimensional kinematic model employing a randomly chosen minimal set of measurements, selected among the ones in a local time window. Next we test the consistency of all the available measurement against the estimated model. This process is repeated for different minimal sets, and the final decision is based on the model for which the greatest number of measurement passed the consistency check.

In standard RANSAC approaches each model is given in a closed form for each minimal set of measurements, for instance, in line fitting the parameters of each line hypothesis are obtained analytically from the coordinates of two points. In our case, complex non-linear models relate sensor readings to hidden model parameters, and it is difficult to give closed form estimators for these parameters for general measurement domains, such as those handled by the ROAMFREE framework. We thus employ the Levenberg/Marquardt optimization algorithm to determine model parameters for each minimal set. The low dimensionality of the employed model ensure fast and reliable convergence.

The proposed mechanism is able to exploit the redundancy available in arbitrary, etherogeneous, asynchronous, sensor readings, effectively overcoming the homogeneity limitations that arises in simple consensus

schemes. Moreover, it does not rely on the current state estimate and it is able to discover coherent set of measurements *before* they are employed for state estimation.

In robotics, the RANSAC paradigm is widely employed in many applications. For instance, it is common to employ such schema to robustify the feature association in visual odometry systems. This method has also been proposed to remove outlier loop-closure constraints in SLAM algorithms, even though there has been some criticism [58] related to the fact that, being RANSAC a randomized algorithm, it might fail to detect *all* the outliers, even though the authors acknowledge that it might be employed as a fast way to remove the most of them. However, we argue that this features are *pros* in our context, since we aim at real-time operation and, with respect to the measurement domain we consider, a spurious outlier seldom has not the dramatic effect of a false loop closure constraint in SLAM.

5.1. A RANSAC Approach for Outlier Detection

In this work we employ a low dimensional, kinematic model to approximate the robot trajectory in a local time window. This model is characterized by a limited number of parameters that are estimated from a minimal subset of the available sensor readings. Multiple hypotheses for the parameter values are formulated employing different, randomly selected, training sets. Heuristics are employed, depending on the measurement domain, to guide the sampling, ensuring that the minimal sets contain enough information such that all the model parameters are observable.

For each minimal set of measurements, $\mathcal{Z}_{\text{fit}}$, the values for the model parameters are obtained by means of a non-linear least squares optimization employing the Levenberg-Marquardt algorithm. The kinematic model plays a role analogous to the backward augmented state estimator. Indeed, given the values for the model parameters, Θ, the extended state $\hat{x}_S(t, \Theta)$ can be computed, as we will discuss in Section 5.2 and the error models introduced in Section 4.3 can be evaluated as a function of the kinematic model parameters.

For each model hypothesis, the residuals for the remaining available readings, $\mathcal{Z} \setminus \mathcal{Z}_{\text{fit}}$, are evaluated and a χ^2 test is performed, allowing us to classify them as inliers or outliers, according to the current hypothesis. The number of model hypotheses k that have to be generated can be determined as a function of the desired probability of identifying an outlier free measurement set, ρ, the inlier probability P_i and the

cardinality of Z_{fit}, N, by means of the well known formula:

$$k = \frac{\log(1-\rho)}{\log(1-P_i^N)} \tag{5.1}$$

where the strength of the RANSAC approach appears. In typical cases few hypothesis have to be generated; for instance, consider $\rho = 0.99$, $w = 0.7$, and $N = 3$, we have to repeat the model fit process only eleven times to be reasonably sure that we have sampled at least an outlier-free measurement subset.

An estimate of the inlier probability is maintained counting the number of inliers in the final model hypothesis, and we iterate through the model fit and measurement validation steps until the probability of having sampled an outlier-free training set exceeded a given threshold. Finally, the model hypothesis for which the fewest sensor readings were flagged as outliers is chosen.

The complete algorithm is reported below, where $\mathcal{H}$ is the set of the generated hypotheses, $\mathcal{Z}$ is composed by all the input sensor readings, $\mathcal{Z}_{\text{fit}}$ is the minimal set employed for parameter estimation and $\mathcal{O}$ is the set of the outliers with respect to the current trajectory model.

Algorithm 2 Outlier rejection

```
n ← 0, H ← ∅
while n < log(1−ρ) / log(1−P_i^N) do
    Z_fit ← ChoseReadings()
    Θ ← FitTrajectoryModel(Z_fit)

    O ← ∅                                   ▷ test all the remaining readings
    for all z ∈ Z \ Z_fit do
        e ← ẑ(t, x̂_S(Θ)) ⊟ z                ▷ evaluate the prediction error
        if e Σ_z^{-1} e^T > λ then          ▷ χ² test
            O ← O ∪ z                       ▷ z is an outlier
        end if
    end for
    H ← H ∪ {Z_fit, Θ, O}                   ▷ store current hypothesis
    n ← n + 1
end while
h ← argmin_{h∈H} #(O)                       ▷ Chose the hypothesis with fewer outliers
```

5.2. Kinematic Model

In this section we discuss in detail the low-dimensional kinematic model that we employ to approximate the robot trajectory in a local time window and we show how this can be used to evaluate sensor error models in place of the backward state estimator presented in Section 4.2.

In this work we restrict to a non-holonomic robot and a 2-D motion where the linear velocity vector is always tangent to the robot trajectory. These assumptions are in general satisfied by car-like and differential drive robots, as long as limited side slippage occurs, in which case a non-zero normal component of the velocity might exist in the true robot motion. Conversely, these assumptions do not hold for holonomic vehicles, such as robot with omnidirectional kinematics or MAVs, for which the proposed approach can still be employed once a more expressive kinematic model has been formulated.

According to the proposed model, the absolute 2-D position of the robot with respect to the world fixed frame W, for each t in a local time window, i.e., $t \in [t_0, t_1]$, varies according to the following law:

$$O^{(W)}(t) = \begin{bmatrix} O_0^{(W)} \\ y_0^{(W)} \end{bmatrix} + \int_0^t R\big(\theta_0 + \omega_0 s + \frac{1}{2}\alpha s^2\big) \begin{bmatrix} v_0 + as \\ 0 \end{bmatrix} ds, \quad (5.2)$$

where $R(\theta)$ yields a 2-D rotation matrix, taking vectors from frame O to W, as a function of the 2-D robot bearing θ. Here the model parameters Θ appear: the initial robot position with respect to W, $O_0^{(W)}$, the robot orientation and angular velocity at time t_0, θ_0 and ω_0, the linear and angular accelerations, constant for $t \in [t_0, t_1]$, a and α. Thus $\Theta = [O_{x,0}^{(W)}, O_{y,0}^{(W)}, v_0, \omega_0, a, \alpha]$ and $\Theta \in \mathbb{R}^6$. It is possible to see that the robot velocity, changing linearly with t, is integrated, taking into account the change in orientation of the velocity vector, which is given by a quadratic form in t.

At this point we take advantage of the decoupling between the representation of the robot state and sensor models introduced in Section 3.3. Indeed, in the pose tracking problem the robot state is maintained over a sliding time window and it is composed by a set of transformations $\Gamma_O^W(t)$. Based on this transformations, for each sensor and for each reading we evaluate the backward augmented state estimator, which abstracts from this representation and computes an extended state at the measurement timestamp t, i.e.,

$$\hat{x}_S(t) = \left[S_t^{(W)}, \mathrm{R}_{S_t}^W, v^{(S_t)}, \omega^{(S_t)}, a^{(S_t)}, \alpha^{(S_t)}\right]. \quad (5.3)$$

In outlier rejection, the usual representation of the robot state by means of successive transformations is replaced by the parameters Θ of the low-dimensional kinematic model introduced in Equation 5.2. Once new expressions for $\hat{x}_S(t)$ have been formulated, the very same error models discussed in the previous chapter can be evaluated. Ultimately, we are "plugging" logical sensors on different implementations of the abstract sensors and graph layers.

The new formulation for $\hat{x}_S(t)$, for each sensor S and for each sensor reading timestamp t can be derived from (5.2), remembering that the 2-D assumption does not hold at logical sensor level:

$$S_t^{(W)} = O^{(W)}(t) + R_z\big(\theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2\big)\mathbf{S}^{(W)}, \tag{5.4}$$

$$\mathrm{R}_{S_t}^{W} = R_z\big(\theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2\big)\mathbf{R}_S^O, \tag{5.5}$$

$$v^{(S_t)} = (\mathbf{R}_S^O)^{-1}\left([v_0 + at, 0, 0]^T + [0, 0, \omega_0 + \alpha t]^T \times \mathbf{S}^{(O)}\right), \tag{5.6}$$

$$\omega^{(S_t)} = (\mathbf{R}_S^O)^{-1}\,[0, 0, \omega_0 + \alpha t]^T\,, \tag{5.7}$$

$$\begin{aligned} a^{(S_t)} = (\mathbf{R}_S^O)^{-1}\Big([a, 0, 0]^T + [0, 0, \omega_0 + \alpha t]^T \times [0, 0, \omega_0 + \alpha t]^T \times \mathbf{S}^{(O)} + \\ + [0, 0, \alpha]^T \times \mathbf{S}^{(O)}\Big)\,, \end{aligned} \tag{5.8}$$

$$\alpha^{(S_t)} = (\mathbf{R}_S^O)^{-1}\,[0, 0, \alpha]^T\,. \tag{5.9}$$

In these equations, $R_z(\theta)$ gives the 3-D rotation matrix around the z axis of the global frame. Note that the calibration parameters $\mathbf{S}^{(O)}$ and $\mathbf{R}_S^O$ are still employed to represent the possible non-null transformation from O to S. As a final remark, note that in (5.4) the absolute robot pose $O^{(W)}(t)$ appears, even though we do not have an analytic, closed form, solution for the definite integral in (5.2). However, if the integrand function is substituted with its Taylor expansion, an approximate solution for it can be obtained in closed form, so that no iterative, numerical, integration algorithm has to be employed.

Now it is possible to relate an arbitrary sensor reading to the underlying estimation variables, which in this case are constituted by the kinematic model parameters Θ, as opposed to the usual, pose-tracking, case in which the robot state is represented by successive transformations between frame W and frame O. This is done by means of the logical sensor error models defined in Section 3.3 and by the new expressions for $\hat{x}_S(t)$ as in (5.4) to (5.9). Once a minimal set of sensor observations $\mathcal{Z}_{\text{fit}}$ is sampled such that each parameter is observable, a Levenberg-Marquardt algorithm can be employed to retrieve the model parameters. Here abstract and logical sensor calibration parameters are

assumed to be known and time-invariant.

5.3. Sampling the Minimal Sets

In this section we discuss the heuristics needed to build the sets $\mathcal{Z}_{\text{fit}}$, ensuring that the resulting optimization problem is not under-constrained.

To estimate the kinematic model parameters Θ we have to pick a minimal set of observations among multiple, heterogeneous, asynchronous information streams. It is important to note that the cardinality of these sets, N, is critical. In particular, N has to be kept as low as possible since the number of model hypotheses that have to be evaluated, k, increases exponentially with N, (see Equation 5.1). In our case N is not fixed and depend on the type of the observations that falls in $\mathcal{Z}_{\text{fit}}$.

Let us start from the straightforward observation that the set $\mathcal{Z}_{\text{fit}}$ cannot be composed by an arbitrary assortment of sensor readings. For instance, any number of linear velocity readings will never make the $O_{x,0}^{(W)}$ and $O_{y,0}^{(W)}$ parameters observable. However, we might not need to estimate these part of the state if all the error models for the elements in $\mathcal{Z}$ do not involve the $S_t^{(W)}$ component of $\hat{x}_S(t)$. Indeed, if the set of available readings contains only relative measurements, such as linear/angular velocity, or acceleration readings, the absolute position of the robot at time t_0 cannot be determined, but it is also not needed to evaluate error models or to detect outliers.

The composition of $\mathcal{Z}_{\text{fit}}$ has to be determined such that any of the model parameters Θ appearing in the components of $\hat{x}_S(t)$ that are employed in sensor models for remaining readings $\mathcal{Z} \setminus \mathcal{Z}_{\text{fit}}$, has to be observable.

To fill the $\mathcal{Z}_{\text{fit}}$ set we employ the following heuristic that has worked in a preliminary evaluation, even though we have not undertaken a rigorous observability analysis yet: first we pick at random a small number of measurements, next we examine which components of the extended state are required by all the sensor models in $\mathcal{Z}$ and add further edges to $\mathcal{Z}_{\text{fit}}$, this time guiding the sampling towards certain measurement types. For instance, if the $O_{x,0}^{(W)}$, $O_{y,0}^{(W)}$ or θ_0 parameter need to be estimated, we sample among error models that depend on $S_t^{(W)}$ and/or $\mathrm{R}_{S_t}^{W}$, until two absolute position constraints or one position and one orientation constraint are in $\mathcal{Z}_{\text{fit}}$.

It is possible to see that, because of the aforesaid heuristic, the number of measurements in the minimal sets is not fixed and it depends on the available sensors, on the relative rates of operation and on the random sampling. We thus maintain an estimate of the average cardinality of

$\mathcal{Z}_{\text{fit}}$ so that Equation 5.1 can be evaluated and a stopping condition for the algorithm can be formulated.

5.4. Experimental Evaluation

The proposed approach is still subject of active research and, even though a preliminary evaluation on synthetic data has confirmed feasibility of the approach, comprehensive benchmarks on real world data are still ongoing and will appear on the final version of this manuscript.

5.5. Conclusions and Open Issues

In this chapter we have proposed an original formulation of an outlier rejection heuristic to be applied along with the ROAMFREE framework to the problem of pose-tracking by means of multi-sensor fusion.

However, a number of critical aspects with respect to the proposed approach would require further analysis. First, we employ a kinematic model to approximate the robot motion in a local time window. However, this model is based on tight assumptions such as non-holonomic motion, which were introduced to reduce the dimension of the state space. Because of these assumptions the model is not able to describe every possible robot motion, even in the specific cases for which it has been designed for. For instance, in RANSAC approaches to line fitting, each model hypothesis exactly pass through the given example points. In our case, the true robot motion will never be *exaclty* the one described by the model. But how do we quantify this discrepancy in the worst case? Isn't it possible that for a certain robot motion the best model that can be found still classifies good measurements as outliers? In other words, there might be cases for which measurements are outlier with respect to any model in the considered class, but are inlier with respect to the true motion.

Another issue lies in the fact that we are employing an iterative optimization algorithm to perform model fitting. Due to the non-linear nature of the problem, the optimization might converge to a local minimum of the error functions associated to the observations in $\mathcal{Z}_{\text{fit}}$. Moreover, the final estimate, in cases of multiple attractors, might depend on the initial guesses for the model parameters. Instead, going back to the line fitting problem, give two example points, the line passing through them is given in a closed form. How do we assess if the model fit procedure has converged to the correct minima? However, this issue might not be critical in the sense that such a model hypothesis will likely classify as

outliers most of the observation in $\mathcal{Z}$, and thus it will be most probably discarded. But what if no model can be found that fits at least the elements in the training set?

An in-depth analysis of these issues, along with a comprehensive experimental evaluation of the proposed approach, are currently ongoing.

Part II.

Experimental Evaluation

Chapter 6

The QUADRIVIO ATV

In this chapter we discuss parameter self-calibration and online pose tracking experiments for the Quadrivio ATV [7], an all-terrain vehicle designed for research in localization, planning, and trajectory control in off-road, rough terrain, environments. In the first part of the chapter we present the robot perceiving architecture and we employ the ROAMFREE framework to perform offline sensor parameter self-calibration. In the remaining part of the chapter, we introduce ROAMROS, a ROS node based on ROAMFREE that performs off-the-shelf pose tracking, and we discuss experiments in which the pose estimates feed the trajectory follower module during autonomous navigation.

6.1. Platform Description

The vehicle considered is a Yamaha Grizzly 700, a commercial fuel powered All-Terrain Vehicle (ATV) equipped with an electric power steering (EPS). It is a utility ATV and it is thus specifically designed for agriculture work. As a result it has a total load capacity of 130 Kg, and it is equipped with a rear tow hook. The main characteristics of the vehicle are listed in Table 6.1.

For the purposes of the project, the original vehicle cover has been removed and substituted with an aluminum one that allows to easily accommodate for the control hardware and the sensors (Figure 6.1). Fur-

Figure 6.1.: The Quadrivio off-road autonomous vehicle.

Main characteristics of the vehicle	
Engine type	686cc, 4-stroke, liquid-cooled, 4 valves
Drive train	2WD, 4WD, locked 4WD
Transmission	V-belt with all-wheel engine braking
Brakes	dual hydraulic disc (both f/r)
Suspensions	independent double wishbone (both f/r)
Steering System	Ackermann
Dimensions (LxWxH)	2.065 x 1.180 x 1.240 m
Weight	296 Kg (empty tank)

Table 6.1.: Vehicle characteristics

thermore, the vehicle has been equipped with three low-level servomechanisms, each one with its own control loop, to automatically regulate the steer position, the throttle aperture and the braking force [6].

The Quadrivio ATV is equipped with the following sensors: a Trimble 5700 GPS receiver, an Xsens MTi inertial measurement unit (IMU), composed by 3D MEMS accelerometer, magnetometer and gyroscope, a four planes Sick LD-MRS laser range-finder mounted on the front top of the chassis, plus a single plane LMS-291 mounted in the front lower part, near the wheels, a stereo rig composed of two Prosilica GC650C cameras, looking sideways, and handlebar and rear wheels encoders. However, at the present stage of development, the laser range-finders and the stereo rig are not employed for pose tracking purposes. Furthermore, the custom aluminum cover has a different mass distribution with respect the the original chassis so that the suspension system fails to soften the vibrations induced by the terrain and the fuel engine. This causes the accelerometer readings to bee too noisy to contribute in pose tracking.

6.2. Parameter Calibration

In the following we discuss how the relevant calibration parameters for the Quadrivio ATV have been retrieved by means of ROAMFREE.

6.2.1. Problem Description

We first review the logical sensors configured and their relevant calibration parameters. Please refer to Section 4.3 for a detailed description of the error models.

An Ackermann kinematic sensor is the architecture master sensor, i.e., the one which triggers new pose nodes to be inserted into the factor graph as new odometry readings are available. Handlebar and rear wheel encoders directly feed this sensor and are read at a frequency of 20 Hz. The kinematic model is characterized by four calibration parameters: $\mathbf{a}_s$ and $\mathbf{b}_s$, which allow to approximate the non-linear relation existing between the handlebar and the actual steering angle in four wheel Ackermann vehicles, $\mathbf{k}_v$, which is a gain parameter for the tangential speed, measured by means of wheel encoders whose number of ticks per revolution might be unknown, and finally $\mathbf{L}$, the distance betweeen front and rear wheel axis.

The GPS readings, after being converted from Longitude-Latitude-Altitude to the East-North-Up coordinate systems, feeds an absolute position logical sensor at a maximum rate of 5 Hz. Here the GPS antenna is mounted at the center of right side of the vehicle chassis, thus the misalignment parameter $\mathbf{S}_{GPS}^{(O)}$ can not be neglected.

The xSens magnetometer drives a vector field logical sensors. Substantial soft and iron distortion effects are present because of the ferromagnetic properties of the vehicle body and the electrical activity in the fuel engine, and are handled by means of the dedicated $\mathbf{A}$ and $\mathbf{b}$ calibration parameters.

Finally, an angular velocity logical sensor is driven by the gyroscope, which we assume to be unbiased and undistorted. The xSens MTI is configured to run at 20 Hz so that it does not exceed the architecture master sensor rate.

Because of the nonlinear nature of the optimization problem, at least rough initial guesses are needed for all the calibration parameters. These are obtained as follows: $\mathbf{a}_s$ is determined from a theoretical analysis of the vehicle steering geometry, from which it results that it should range from 0.49 to 0.56, where the uncertainty comes from the fact that it is difficult to precisely know the dimensions of all the involved steering gears and mechanisms. $\mathbf{k}_v$ is set to 0.1 examining the encoder

integrals after the vehicle has been moved for a known distance. $\mathbf{L}$ is equal to 1.25 m and its value comes from the Yamaha Grizzly 700 documentation. This value is considered correct and it is not subject to estimation. The GPS displacement, $\mathbf{S}_{GPS}^{(O)}$, has beed determined by means of a direct inspection on the vehicle and it is initialized to $[0.70, -0.46, 0.88]^T$ m. The Earth magnetic field $\vec{h}^{(W)}$ is supposed to be known and constant in the robot operation area and its value is set according to the USA National Geophysical Data Center (NGDC)[1] to $[0.007719, 0.225097, -0.416133]$ G. The remaining parameters are impossible or very unpractical to determine directly and are set to default values, i.e., $\mathbf{A} = I_{3\times3}$, $\mathbf{b} = 0_{3\times1}$ and $\mathbf{b}_s = 0$. The error models developed in this work provide further calibration parameters, yet they are not considered because either are not observable, as we will discuss later on, or because their effect on the pose tracking accuracy is marginal.

We perform the parameter calibration by means of an offline optimization over the whole set of sensor readings collected while manually driving the robot along circles of different radii, so that the steering geometry and the magnetometer distortion parameters, as well as the GPS misplacement, are fully excited. To this end, we saturate the handlebar angle setpoints to 15, 30 and 40 degrees, resulting respectively in the g15, g30 and the g40 datasets. Unfortunately, only flat terrain was available for field experiments. The resulting trajectories are depicted in Figure 6.2, 6.3 and 6.4. In the second one it is possible to see that the GPS readings are affected by substantial multi-path effects, i.e., the circumstance in which environmental features cause a combination of reflected and/or diffracted replica signals to arrive at the receiving antenna, resulting in discontinuities in the GPS position estimates.

Notes on Parameter Observability

Before proceeding, we discuss certain limitations which arise when one attempts to determine the full set of parameters considering only 2-D motion. Even though a rigorous observability analysis could be carried out with techniques based on the Lie derivatives [71, Section 6.2], simpler considerations will suffice to show the point.

Before we start, it is important to remark that ROAMFREE is a 6-DOF pose tracking framework, thus, even in case of 2-D motion, we always deal with 6-DoF poses and the estimates will not, in general, lie on a plane, because of magnetometer and GPS altitude noise. Please note that in principle we could arbitrarily enforce planarity in estimates

[1] http://www.ngdc.noaa.gov

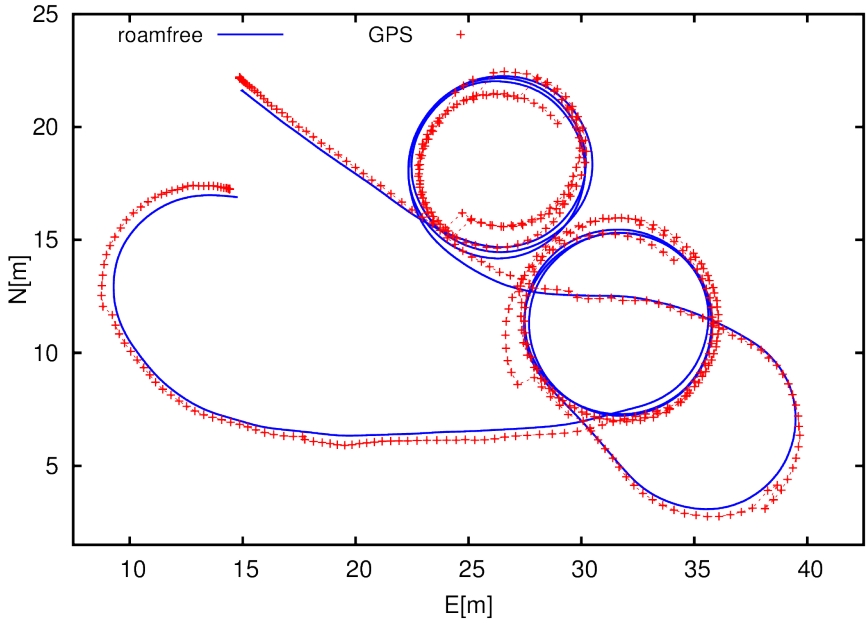


Figure 6.2.: Dataset g15, handlebar setpoints saturation: 15°, with GPS readings and the robot poses obtained after calibration.

simply acting on the Ackermann kinematic sensor noise covariance matrix. Indeed, as it was discussed in Section 4.4, extra constraints augment the kinematic model, i.e., $v_z^{(S)} = 0$, $\omega_x^{(S)} = 0$ and $\omega_y^{(S)} = 0$, with $S \equiv O$, stating that the position of the robot cannot change over the z axis of W, and that the robot orientation cannot change along the roll and pitch axis of the sensor frame.

We start our analysis from the z component of the GPS misalignment parameter: it is possible to see that

$$\begin{aligned} S_{GPS}^{(W)}(t) &= \Gamma_O^W(t)\mathbf{S}_{GPS}^{(O)} = \\ &= R_O^W(t)\left(\mathbf{S}_{GPS}^{(O)} + [0,0,z]^T\right) + \\ &\quad + O^{(W)}(t) - [0,0,z]^T \end{aligned} \tag{6.1}$$

holds if $R_O^W(t)$ is a 2-D rotation for each t, showing that an arbitrary GPS displacement along the robot z axis can be compensated by an opposite translation of all the robot poses.

Next we move to the vector field readings. First recall the error model

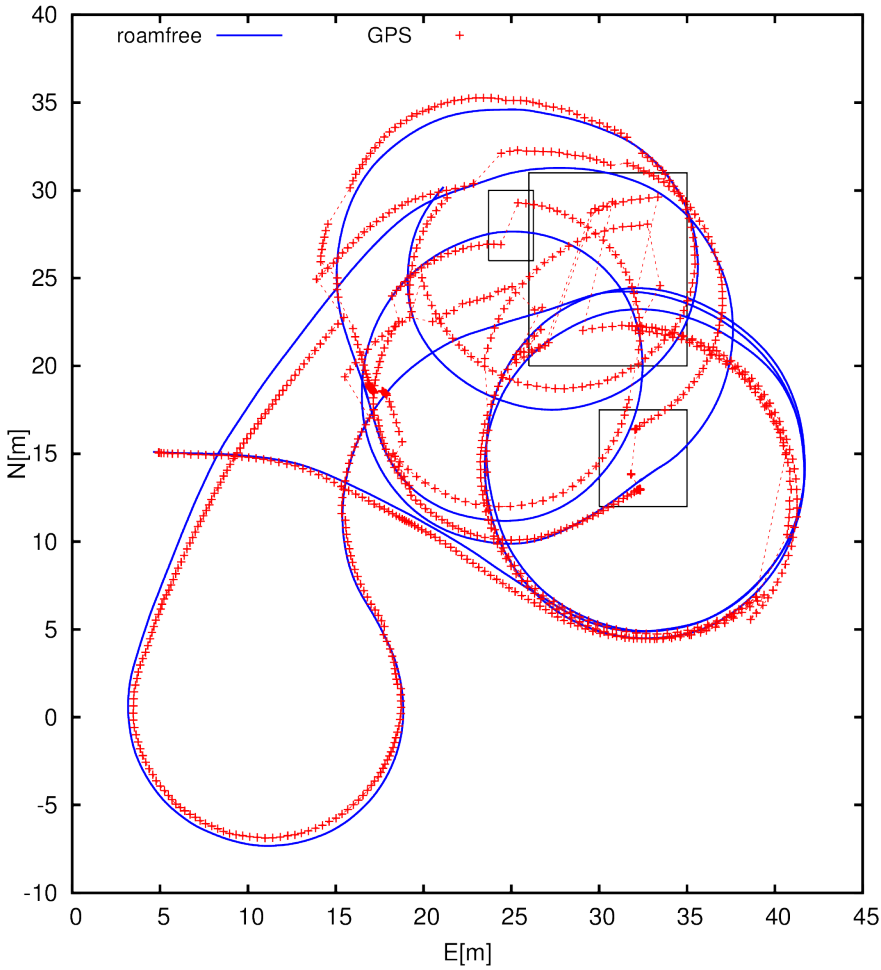


Figure 6.3.: Dataset g30, handlebar setpoints saturation: 30°. The boxes highlight GPS multi-path effect.

as it has been presented in Section 4.3.6:

$$\hat{z}(t) = \vec{h}^{(S)}(t) = \left[\mathbf{A}\left(\mathbf{R}_S^O\right)^{-1}\right] R_O^W(t)^{-1}\vec{h}^{\,(W)} + \mathbf{b}. \tag{6.2}$$

where $\mathbf{A}$ is an unconstrained 3×3 matrix, and thus it can also encode a 3-D rotation. While other sensor provide further information on $R_O^W(t)$, in the considered case we have little information about $\mathbf{R}_S^O$, i.e., the IMU misalignment parameter. In fact, even if the gyroscope sensor it is housed within the xSens MTI and thus shares the misalignment

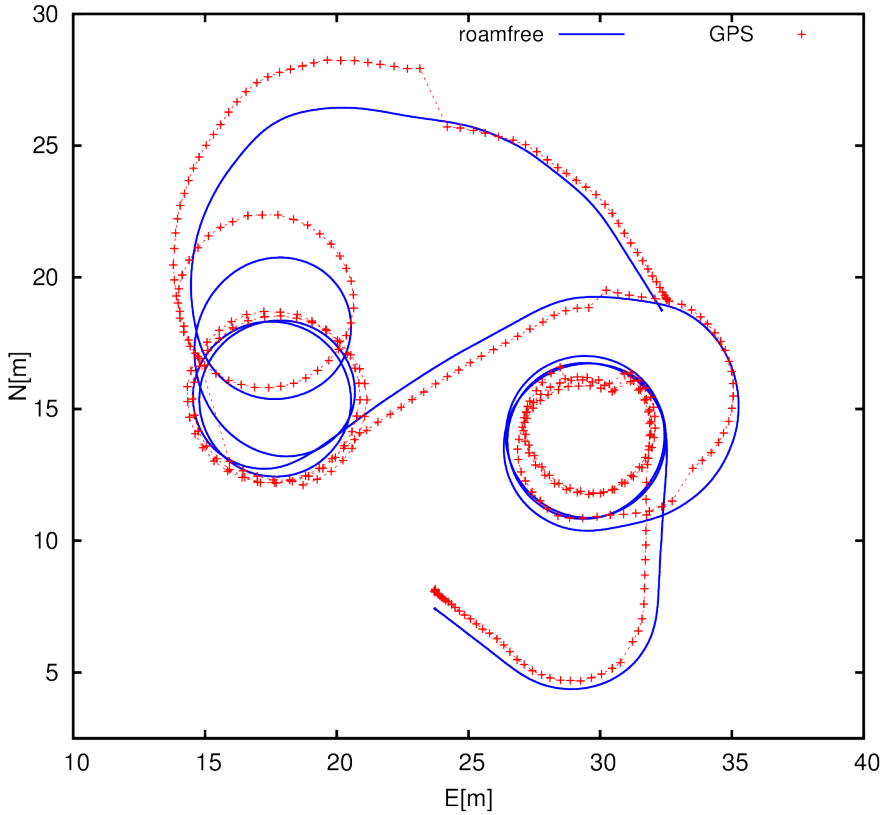


Figure 6.4.: Dataset g40, handlebar setpoints saturation: 40°.

parameter with the magnetometer, its x and y axis are too noisy and can not be trusted enough so that the least-squares solver can disambiguate between the $\mathbf{A}$ and the $\mathbf{R}_S^O$ contribution in (6.2). This fact hold also for 3-D trajectories and it has been already pointed out in [83]. In Quadrivio, we carefully align the IMU with the robot chassis and we consider $\mathbf{R}_S^O = I$.

Consider now the two vector field calibration parameters $\mathbf{A}$ and $\mathbf{b}$.

Again, if $R_O^W(t)$ is a 2-D rotation for each t, it follows that:

$$\begin{aligned}
\vec{h}^{(S)}(t) &= \mathbf{A}R_O^W(t)^{-1}\vec{h}^{\,(W)} + \mathbf{b} = \\
&= \begin{bmatrix} \mathbf{A}_{(1:2,1:2)} & \mathbf{A}_{(1:2,3)} \\ \mathbf{A}_{(3,1:2)} & \mathbf{A}_{(3,3)} \end{bmatrix} \begin{bmatrix} R_O^W(t)^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{h}_{xy}^{(W)} \\ \vec{h}_z^{(W)} \end{bmatrix} + \mathbf{b} = \\
&= \begin{bmatrix} \mathbf{A}_{(1:2,1:2)}R_O^W(t)^{-1}\vec{h}_{xy}^{(W)} \\ \mathbf{A}_{(3,1:2)}R_O^W(t)^{-1}\vec{h}_{xy}^{(W)} \end{bmatrix} + \begin{bmatrix} \mathbf{A}_{(3,1:2)}\vec{h}_z^{(W)} \\ \mathbf{A}_{(3,3)}\vec{h}_z^{(W)} \end{bmatrix} + \mathbf{b}. \qquad (6.3)
\end{aligned}$$

where in the last equation a sum of two unknown quantities appears. If we try to estimate both $\mathbf{A}$ and $\mathbf{b}$ at the same time from a 2-D motion, we introduce a *gauge freedom* of three degrees. In fact, if we add an arbitrary quantity δ to $\mathbf{b}$, no change occurs in $\vec{h}^{(S)}(t)$ if we subtract $\delta/\vec{h}_z^{(W)}$ from $\mathbf{A}_{(1:3,3)}$.

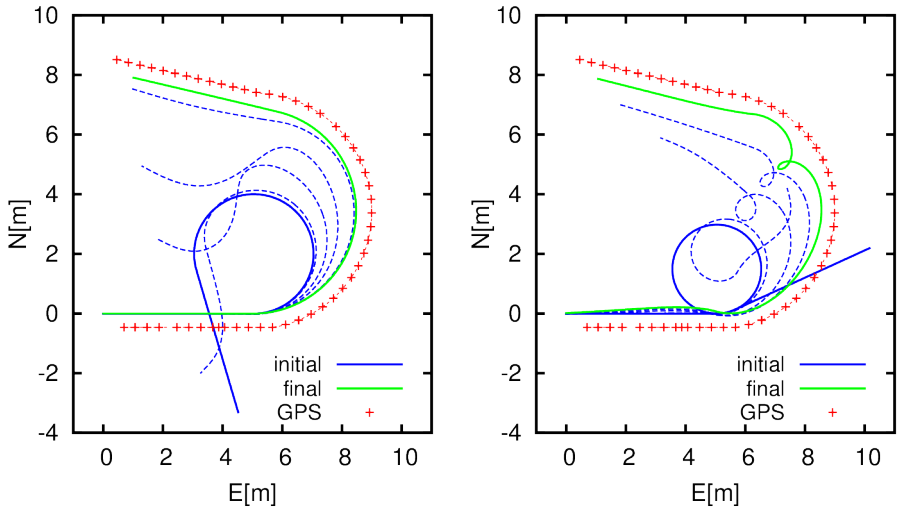
6.2.2. Pose Tangles and Calibration Heuristics

There are cases in which the optimization algorithm might fail to converge to the desired attractor. Here we can not speak in terms of global and local optima since this would require a rigorous characterization of the fitness landscape. Moreover, in the considered experiments the ground truth for the robot trajectory is not available. Nevertheless, it is quite easy for humans to guess the true robot trajectory looking at GPS tracks, even in presence of multi-path effect. Thus, we can visually discern the quality of the solution obtained by least-squares estimation.

As it was discussed in previous sections, because of the non-linearity of the optimization problems we are considering, we need a reasonable initial guess for all the variables in the problem[2]. For the Quadrivio ATV, this role is played by the Ackermann kinematics logical sensor. However, this sensor sports calibration parameters for which we have a rough prior knowledge, i.e., $\mathbf{a}_s$ and $\mathbf{b}_s$. Thus, it might happen that the initial guesses generated for the robot poses are not accurate enough and estimation fails to converge to the proper trajectory.

Here we discuss a synthetic example in which the effects of a bad initialization for the $\mathbf{a}_s$ parameter are explored. We consider a simulated vehicle identical to the Quadrivio ATV. We suppose it moves straight for 5 s at 1 m/s, then it steers with an angle of 20 degrees for 10 s, and finally it moves straight for 5 s more. We drive the ROAMFREE sensor fusion engine with ground truth readings for the GPS and the

[2]Recall from Section 3.2.1 that the architecture master sensor is employed to obtain initial guesses for the new pose nodes, employing the current sensor readings and the forward kinematics equations.



(a) $\mathbf{a}_s = 0.8$, proper convergence.

(b) $\mathbf{a}_s = 1.0$, failure.

Figure 6.5.: Evolution of pose estimates during optimization for two different initial guesses for the $\mathbf{a}_s$ parameter, and consequently for the robot poses, given the same set of sensor readings.

Ackermann kinematics, yet we incorrectly initialize $\mathbf{a}_s$ to 0.8 and 1.0, while it true value is 0.5. Finally we run the estimation by means of the Levenberg-Marquardt algorithm, attempting to estimate the steering gain parameter.

In Figure 6.5a we have plotted the initial guess for the robot poses, the final estimates (in green) and the intermediate pose estimates during the optimization. It is possible to see that even though the initial guess for the robot pose was substantially different with respect to the GPS readings, the retrieved poses match the GPS estimates. Moreover, the correct value for $\mathbf{a}_s$ was retrieved, i.e., $\hat{\mathbf{a}}_s = 0.4995$. On the contrary, in Figure 6.5b, the algorithm was not able to remove the loop existing in the trajectory initial guess. Indeed, to achieve this, a temporary decrease in the likelihood of the robot poses with respect to the sensor readings is required, and this is not allowed by a greedy algorithm such as LM. In this case the estimate for the steer gain was $\hat{\mathbf{a}}_s = 1.2532$, which is not correct.

To handle the issues previously described, we developed the following calibration heuristic, which we have successfully applied to the presented

		g15	g30	g40	avg
$\mathbf{k}_v$		0.060 ± 0.002	0.057 ± 0.003	0.057 ± 0.003	0.058 ± 0.001
$\mathbf{a}_s$		-0.544 ± 0.110	-0.573 ± 0.075	-0.561 ± 0.063	-0.562 ± 0.044
$\mathbf{b}_s$		-0.010 ± 0.026	-0.006 ± 0.034	0.001 ± 0.034	-0.006 ± 0.018
$\mathbf{A}$		$\begin{bmatrix} 1.80 & -0.11 & 0.02 \\ 0.14 & 1.76 & 0.00 \\ 0.09 & -0.03 & 0.64 \end{bmatrix}$	$\begin{bmatrix} 1.93 & -0.06 & 0.01 \\ 0.10 & 1.87 & -0.00 \\ 0.11 & -0.06 & 0.60 \end{bmatrix}$	$\begin{bmatrix} 1.81 & 0.04 & 0.00 \\ -0.02 & 1.75 & -0.02 \\ 0.12 & -0.01 & 0.60 \end{bmatrix}$	$\begin{bmatrix} 1.85 & -0.04 & 0.01 \\ 0.07 & 1.80 & -0.01 \\ 0.11 & -0.03 & 0.61 \end{bmatrix}$
$\mathbf{b}$		$\begin{bmatrix} -0.13 \\ 0.03 \\ -0.50 \end{bmatrix}$	$\begin{bmatrix} -0.13 \\ 0.01 \\ -0.52 \end{bmatrix}$	$\begin{bmatrix} -0.13 \\ 0.03 \\ -0.52 \end{bmatrix}$	$\begin{bmatrix} -0.13 \\ 0.02 \\ -0.51 \end{bmatrix}$
$\mathbf{S}_{GPS}^{(O)}$	x	0.205 ± 0.233	0.385 ± 0.212	0.343 ± 0.227	0.316 ± 0.129
	y	-0.331 ± 0.285	-0.354 ± 0.235	-0.496 ± 0.246	-0.398 ± 0.146

Table 6.2.: Results for the calibration parameters.

(and more) datasets: at the beginning of the estimation the only parameters which are set to be estimated are the Ackermann kinematic ones, $\mathbf{a}_s$, $\mathbf{b}_s$ and $\mathbf{k}_v$. We do this because these are employed to initialize new pose nodes and thus reasonable estimates for these parameters have to be available as soon as possible. We then insert 10 s of sensor readings and run the estimation. We iterate through this process until the all the available readings have been considered. In this way we can obtain initial guesses for all the robot poses even if we only have rough guesses for the kinematic parameters. Next we free the vector field parameter $\mathbf{b}$ and run the estimation again, so that the average magnetic field value is estimated. Note that a bias exist at least for the z component, since 2-D motion is considered. Finally, $\mathbf{b}$ is fixed again so we can estimate the $\mathbf{A}$ parameter. In this final run we also free the remaining parameters, i.e., the GPS misplacement $\mathbf{S}_{GPS}^{(O)}$.

It is important to note that parameter calibration procedures are often tricky and it is very difficult to provide general guidelines or heuristics that can be directly applied to multiple, different, problems. This is one of the reasons why ad-hoc solutions are being proposed for specific calibration scenarios. However, thanks to the general and flexible design of our framework, the calibration heuristic for the Quadrivio ATV could be implemented relying only on the available public API and no change in the internals of the engine was required.

6.2.3. Results

In Table 6.2 we have summarized the parameter estimates for each one of the calibration datasets together with their uncertainty weighted averages.

We can evaluate the accuracy of $\mathbf{k}_v$, $\mathbf{a}_s$ and $\mathbf{b}_s$ by integrating odom-

etry readings, first employing their initial guesses and then the values obtained averaging the results obtained for the three training datasets. To this end, we employ a fourth dataset not employed in the calibration runs. The results are shown in Figure 6.6, where a trajectory of ≈ 127 m is considered. It is possible to see that the odometry integral computed with the initial guesses for the kinematic parameters soon diverges from the GPS readings, whereas the calibrated parameters yield accurate dead reckoning, with a final pose distance of around 10 m.

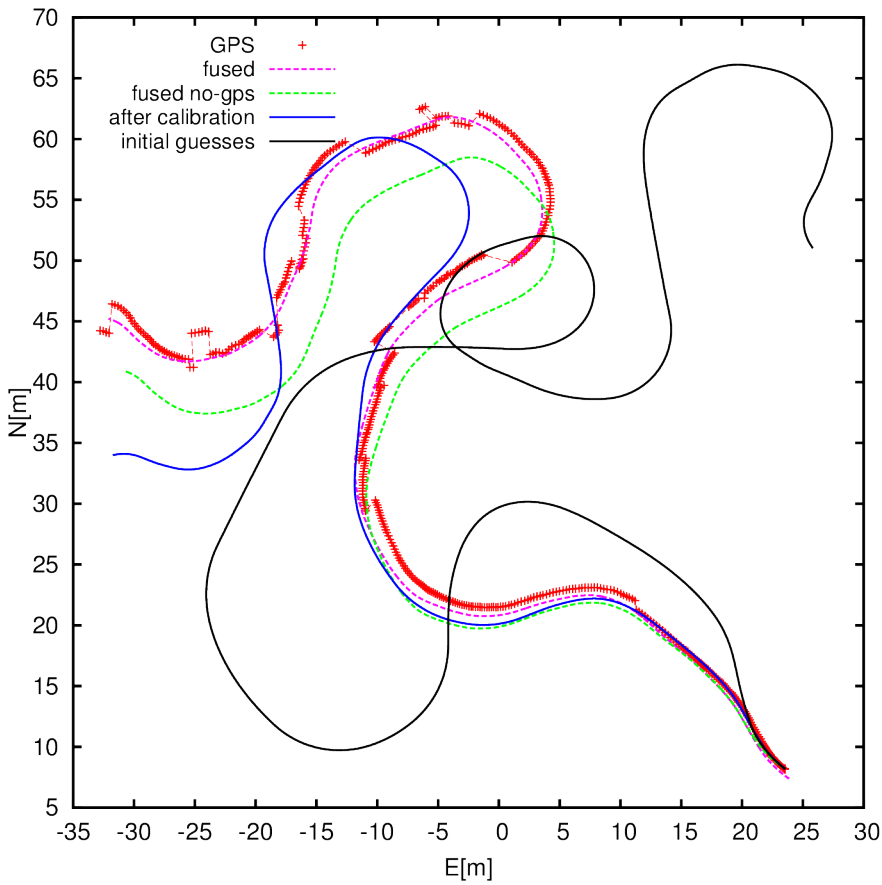
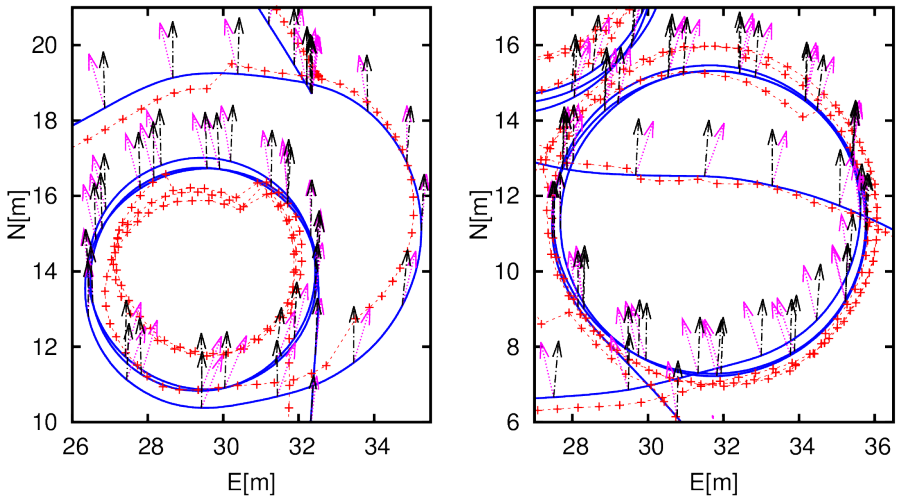


Figure 6.6.: Comparison of odometry integrals computed with initial guesses and calibrated kinematic parameters with respect to the GPS readings and the estimation performed with all of the available sensors.



(a) Dataset g40

(b) Dataset g30

Figure 6.7.: Two portions of the calibration dataset. The magenta arrows show the direction of the Earth magnetic field as perceived by the xSens MTI, while the black ones are corrected for bias and distortion according to $\mathbf{A}$ and $\mathbf{b}$.

We next discuss the results for the magnetometer calibration parameters $\mathbf{A}$ and $\mathbf{b}$. Once the robot orientations $R_O^W(t)$ have been estimated, it is possible to express the magnetic field reading $z(t) = \vec{h}^{(S)}(t)$, which is expressed in the sensor frame S, in the fixed frame W. In Figure 6.7 we have plotted two portions of the estimated trajectory after parameter calibration. Here, the arrows represent the direction of $\vec{h}^{(W)(t)}$ as computed from the magnetic field reading and the robot pose at time t. In particular, magenta arrows are obtained assuming the magnetic field reading is unbiased and undistorted, i.e., $\mathbf{A} = I_{3\times3}$ and $\mathbf{b} = 0_{(3\times1)}$. It is possible to see that in this case a different direction is obtained for different poses, depending on the robot orientation. Conversely, the black arrows are obtained employing the calibrated values for $\mathbf{A}$ and $\mathbf{b}$, showing that they are able to compensate for the bias and distortion affecting magnetometer readings. Indeed, all the arrows consistently point towards the North.

Regarding the remaining calibration parameter, $\mathbf{S}_{GPS}^{(O)}$, the results are less satisfactory. As discussed before, a prior estimate for this parameter is obtained by direct inspection on the vehicle. Even though it is quite

impractical to precisely measure distances between the GPS antenna and the origin of O, placed at the middle point of the rear wheel axis, it is is very unlikely that the initial guess differ from the true values by more than few centimeters. Conversely, the confidence intervals reported by the calibration algorithm are of the order of tens of centimeters (see Table 6.2). Moreover, while the estimated mean value for the displacement along the y axis it is consistent with the direct inspection, this does not hold for the x component, which differs by about 39 cm with respect to the measured value.

We can give a possible explanation of this fact based on GPS latency. First consider that, on the Quadrivio ATV, a regular PC associates a timestamp to each GPS reading as it receives the NMEA sentences from the serial-to-USB channel. Before this, the raw GPS signal has to be processed to obtain LLA estimates. So there is a certain, non-null, latency between the moment in which the GPS signal is available at the receiver and when the ENU coordinates are timestamped at the PC running ROAMFREE and/or logging the data, during which the vehicle might be moving. Thus we are ultimately estimating the GPS displacement with respect to its position at the moment the ENU readings are timestamped. The Trimble 5700 specifications[3] report a processing latency of 0.02 s. Plus, we have to consider the elapsed time during serial transfer of the GGA and GST NMEA sentences, which can be considered to be 120 characters long on an average, resulting in additional ≈ 0.02 s. Furthermore we have to add USB and operating system latencies which are more difficult to quantify. In the considered datasets we were driving the Quadrivio ATV at an average speed of 3 m/s: being 0.04 s a lower bound for the GPS latency, we can expect that the observed value for the x component of $\mathbf{S}_{GPS}^{(O)}$ by means of self-calibration will be lower with respect of the true value by at least 12 cm.

However, the obtained confidence intervals for the GPS displacement are not tighter than the ones for direct inspection, implying that few information can be obtained from other sensors on this parameter, thus, at least in this sensor configuration, we prefer to employ direct measurements instead of the estimated obtained by self-calibration.

6.3. Online Pose Tracking

After having obtained an estimate for the calibration parameters which are impractical or impossible to determine by means of a direct inspection, such as the $\mathbf{A}$ and $\mathbf{b}$ magnetometer distortion coefficients, we de-

[3] http://www.geoplane.com/trimble/pdfs/5700specs.pdf

ploy a pose tracking application based on ROAMFREE whose estimates will be employed in a control loop to follow a given trajectory.

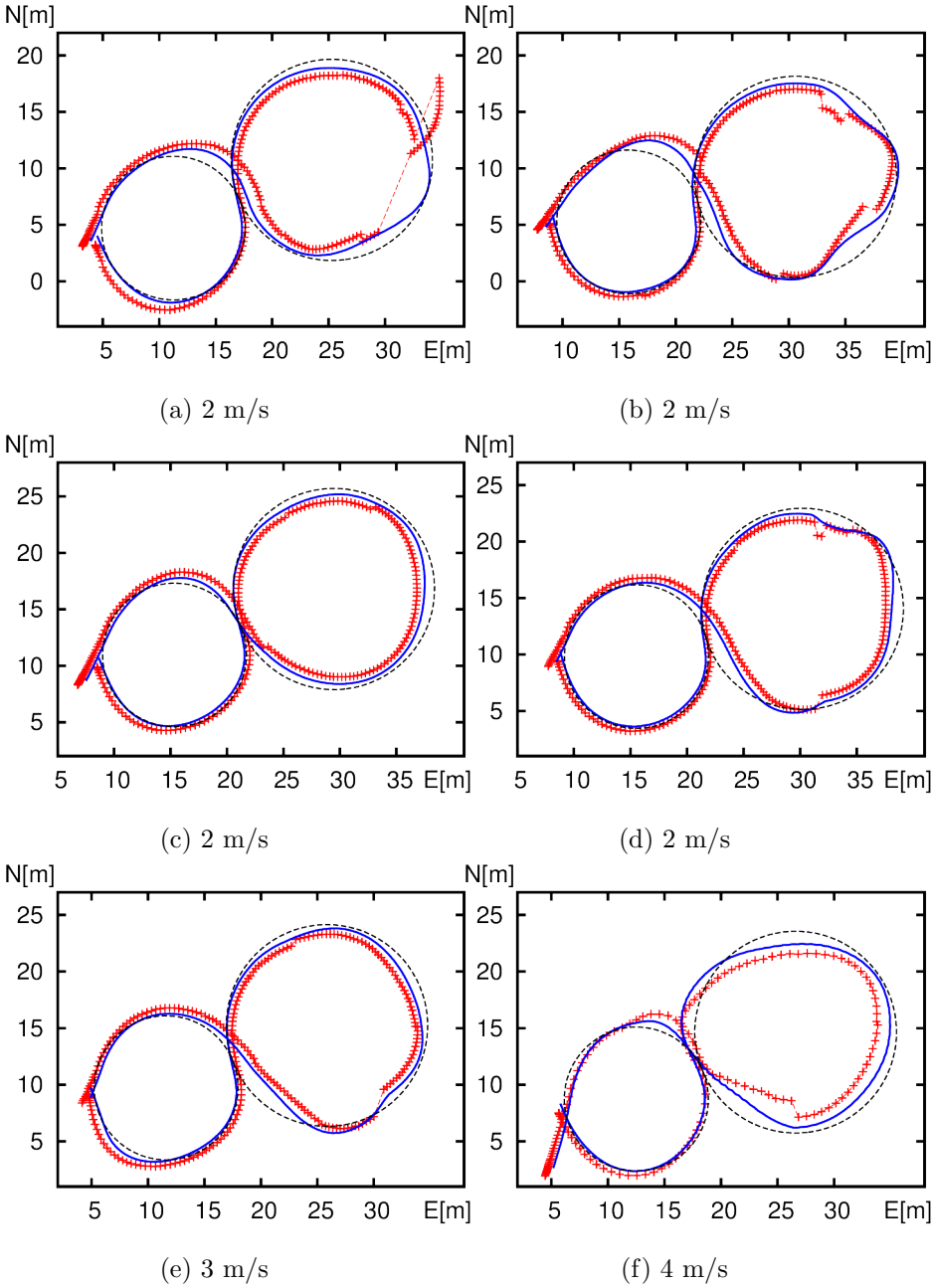
In the previous section we described a calibration heuristic which was implemented as a standalone application employing the ROAMFREE public API. Indeed, a observability analysis showed that the set of parameters subject to simultaneous estimation can not be arbitrary and thus forced us to write a custom application to target the distinctive traits of the considered case. Conversely, once the logical sensor parameters are known, a solution for the online pose tracking problem can be obtained employing ROAMROS, a general purpose ROS node built on the top of the ROAMFREE public API to provide off-the-shelf pose tracking capabilities. It is based on configuration files and allows the end user to configure the needed logical sensors, the required calibration parameters and the ROS topics on which sensor readings are available as standard messages. Moreover, the user can specify the fixed-lag length and the desired rate at which estimation has to be performed, along with other properties of the solver.

In the following experiments, the control node receives the input trajectory as a 2-D spline and it implements a control law, specifically designed for Ackermann steering vehicles, that computes the handlebar position and the throttle setpoints as a function of the current robot pose, obtained by the ROAMROS node.

We generated eight-shaped trajectories that originate 1 m ahead with respect to the current pose of the robot and we trigger autonomous trajectory following. The two circles have a radius of respectively 18 and 12.5 m. In Figure 6.8 we have plotted the reference path, the ROAMFREE position output, and the GPS readings for six experiments where in four of them the speed setpoint was 2 m/s, increasing to 3 and 4 m/s in the last two. We considered 2.5 s of fixed-lag length and we run the estimation at a rate of 20 Hz.

Here we do not aim at benchmarking the motion controller, not at evaluating the adherence of the robot trajectory with the reference one, also because no ground truth position is available. Conversely, we want to show that the ROAMFREE pose tracking capabilities can be successfully employed in motion control loops to increase robustness and accuracy. Consider for instance the experiment depicted in Figure 6.8a: it is possible to see that multi-path effect seriously compromised the GPS position estimates while moving along the big circle part of the trajectory. Nevertheless, this did not affect the pose estimates thanks to the other sensors available and the robot path did not diverge from the given reference. Indeed, after multi-path effect has disappeared in the next part of the trajectory, it is possible to see that the GPS positions

and the roamfree estimates are very adherent with respect to the desired path.



(a) 2 m/s (b) 2 m/s

(c) 2 m/s (d) 2 m/s

(e) 3 m/s (f) 4 m/s

Figure 6.8.: Online trajectory following results. Reference path for the trajectory follower (black dashed line), the ROAMFREE position output (blue line), and the GPS readings (red crosses).

Chapter 7

The LURCH Autonomous Wheelchair

In this chapter we discuss self-calibration benchmarks for the LURCH Autonomous Wheelchair [8] and we compare the results, where possible, with the ones obtained by the ad-hoc algorithm proposed in [19].

7.1. Experiment Description

LURCH (see Figure 7.1) is an robotic wheelchair designed to assist disabled persons by autonomously navigating indoor environments or by performing automatic hazard avoidance during manual drive.

The robot is equipped with multiple, heterogeneous sensors: two URG-04LX laser range-finders, which have a 240 degrees field of view, a 5.6 m range and 10 Hz operating frequency, are mounted such that each one covers one side of the robot plus a portion of the front. These are employed to compute odometry and to detect dynamic and static obstacles in autonomous navigation. The rear direction is instead covered by multiple sonars. Moreover, a Prosilica GC1020 is mounted behind the seatback and looks backward, capturing frames at 10 Hz. Finally, wheel speed is estimated from low resolution encoders at 50 Hz.

While wheel encoder sensor readings can directly feed a differential drive kinematic logical sensor (see Section 4.4.1), the laser range-finder

Figure 7.1.: The LURCH autonomous wheelchair. The two Hokuyo URG-04LX are visible near the footrests, while the Prosilica is mounted behind the seatback.

point clouds and the camera frames are not directly handled by the framework. However, we can apply pre-processing so that one of the handled measurement domain is obtained. In particular, 2D displacement estimates are obtained by matching successive laser scans with an Iterative Closest/Corresponding Point (ICP) algorithm [17].

Note that we do not consider loop closures, i.e., we never align the current scan with a past one in case prior pose knowledge suggests that the robot has came back in the same place, as it happens in graph-SLAM approaches such as [37]. Instead we employ the laser range-finders as redundant odometers to be integrated with the estimates obtained by applying forward kinematics to the wheel encoder outputs.

To fix the absolute position of the robot, we employ the camera to track fiducial markers present in the environment. In this work the we employ the ALVAR library [70] to process the Prosilica frames and compute the position of the visible fiducial markers with respect to the camera, which we then employ to feed multiple landmark position logical sensors, one for each marker (see Section 4.3.7). New logical sensors are added at run-time as additional fiducial markers are detected. In other words, when a new fiducial marker is first seen in a camera frame, the set of sensors is augmented and a new parameter, the absolute position of that marker, $L_i^{(W)}$, is inserted into the problem-graph.

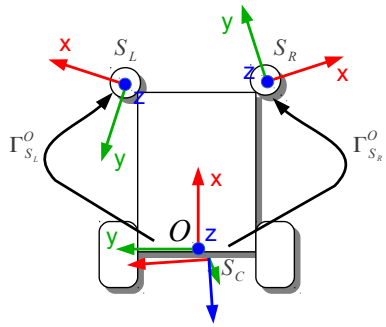


Figure 7.2.: The LURCH sensor frames: S_L and S_R placed at the two Hokuyo range-fiders, and S_C at the Prosilica.

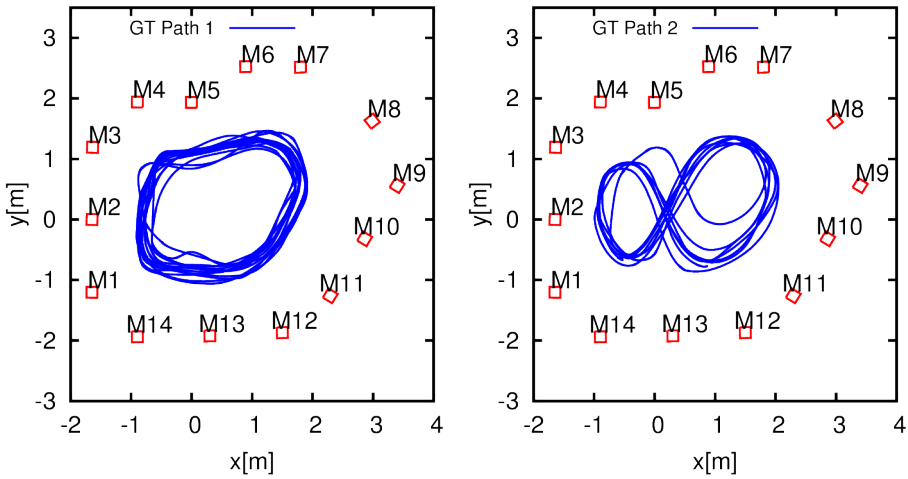
Note that the absolute positions of the landmarks are unknown, thus we are actually solving a SLAM problem in which the goal is to determine the pose of the robot and the position of a set of world-fixed features, which form the map of the world. However, in our case the problem is simpler since each fiducial marker encodes a unique identifier that allows to distinguish it among the others, substantially eliminating the data association problem. Moreover, thanks to the available a-priori information on the marker dimensions, their position and orientation with respect to the camera can be determined on a metric scale looking to a single frame only.

The resulting set of logical sensor available for pose tracking is made up by one differential drive sensor, which handles wheel speed readings, two displacement sensors, one to handle the output of the ICP algorithm for the left range-finder, and one for the right one, and multiple landmark position sensors, one for each detected fiducial marker. In Figure 7.2 we have depicted the associated reference frames. These sensors sport multiple calibration parameters and most of them are substantially misplaced and/or misaligned with respect to the odometric reference frame O. To summarize, the relevant calibration parameters are: the 2D transformation from O to the laser frames S_R and S_L, the 3D camera misalignment $\mathbf{R}^O_{S_C}$, the z component of the camera displacement, the wheel radius $\mathbf{r}$ and the baseline $\mathbf{L}$. Additionally, every landmark logical sensor sports a parameter specifying its own fiducial marker position with respect to W, $\mathbf{L}_i^{(W)}$, which has to be determined at run-time, as it happens in SLAM algorithms. These parameters are critical for the pose tracking accuracy. While direct inspection on the robot might yields acceptable values for some of them, such as for the laser displacements, the others remain difficult to determine in this way.

Figure 7.3.: A bird's eye view of the calibration setup.

In the following we show how the ROAMFREE framework can be employed to refine the rough estimates available for the above sensor parameters. We first setup a benchmark environment in which an area of approximately 25 m^2 is considered and 14 fiducial markers of size 18×18 cm are placed on the floor such that they completely surround the area, as it is depicted in Figure 7.3. The area is covered by an Optitrack motion capture system, which reaches sub-centimeter accuracy and provides us with the ground truth reference for both the wheelchair and the fiducial markers position. Note that the ground truth data is never employed in the calibration process. We collect sensor readings for the subsequent offline estimation runs by manually driving the robot along two kind of paths, depicted in Figure 7.4. The resulting dataset are finally divided in seven slices of 60 s each, with 20 s overlap.

The parameter calibration is done offline. The differential drive logical sensor provide readings at 50 Hz and it is the architecture master sensor. The other three logical sensors have to rely on the poses generated as new odometry readings are available. Both the range finders and the camera produce readings at 10 Hz. For each couple of scans a displacement reading is inserted, while the number of landmark position constraints for each frame clearly depend on the number of visible markers. Since no absolute position information source is available, the robot and the landmark poses can be determine up to a rigid transformation. In other words, in the resulting optimization problem there is a *gauge freedom* of six dimensions. Thus, we set the first pose estimate such that $O \equiv W$ and we fix the related graph vertex. Also note that it is not possible to estimate the z component of the camera misplacement from a 2D motion as long as prior information on the position of the marker is not



(a) Datasets k1-1, ..., k1-5

(b) Datasets k2-1, k1-5

Figure 7.4.: The ground truth for the trajectories employed.

provided in the optimization problem. To clarify this point, note that:

$$\begin{aligned}\mathbf{L}^{(W)} + [0,0,z]^T &= \Gamma_O^W(t)\Gamma_S^O \overbrace{L^{(S)}(t)}^{z(t)} \\ &= R_O^W(t)\mathbf{R}_S^O F^{(S)}(t) \\ &\quad + R_O^W(t)\left(\mathbf{S}^{(O)} + [0,0,z]^T\right) + O^{(W)}. \qquad (7.1)\end{aligned}$$

In case of 2D motion, the rotation $R_O^W(t)$ does not affect an arbitrary offset on the z component of the sensor misalignment parameter $\mathbf{S}^{(O)}$, which can thus be canceled offsetting the landmark position parameter $\mathbf{L}_i^{(W)}$ by the same amount. Thus, if we tried to estimate also the z component of the camera misalignment parameter, we would have introduced another degree of freedom in the optimization problem. This degree of freedom could have be removed introducing prior constraints on the z component of the landmark position parameters, encoding, for instance, the fact that we known that they are placed on the floor.

Since the optimization problem is ultimately a weighted least-squares, the solution heavily depends on the weights, i.e., the covariance matrices associated to each measurement edge. Thus, one of the prerequisite for obtaining accurate and consistent estimates for the calibration parameters and for the robot poses is to have a reasonable estimate for the sensor reading uncertainty.

For what concerns wheel odometry, we assume that the robot motion

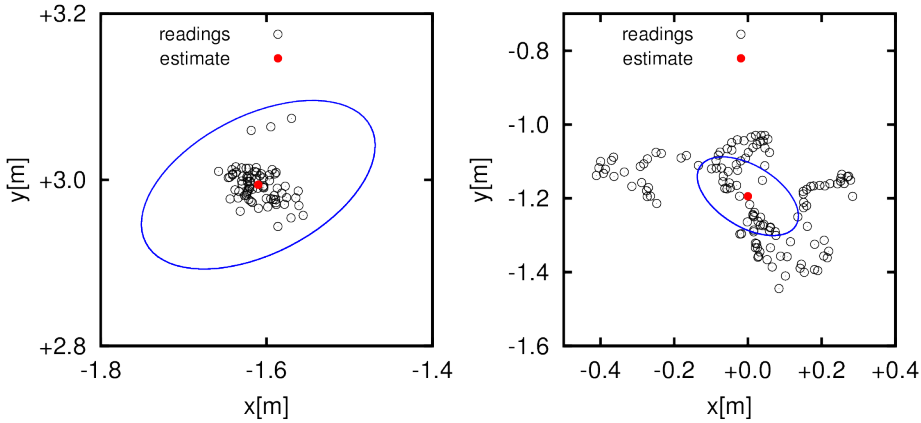
is planar and that slippage in the normal direction with respect to the trajectory is very unlikely to occur. These assumptions are reflected by low variances for the noises on the $v_y^{(S)}$, $v_z^{(S)}$, $\omega_x^{(S)}$ and $\omega_y^{(S)}$ components of the sensor linear velocity, see Section 4.4. The wheel speed estimator, given the encoder ticks per period, is unbiased and uniformly distributed. Thus, given that the LURCH encoders have 180 ticks per wheel revolution and that the sampling frequency is 50 Hz, the variance of the wheel speed estimator needed in the differential drive logical sensor is given by

$$\sigma_\omega^2 = \frac{1}{12}\left(\frac{f_s \cdot 2\pi}{\#\text{ticks}}\right) \simeq 0.25 \text{ rad}^2/\text{s}^2. \tag{7.2}$$

For the other logical sensors, the situation is more difficult: the scan-matching algorithm associates a covariance matrix to each displacement estimate based on the analysis of the error function being minimized in the ICP algorithm. However, this estimate depends on the assumed uncertainty for each distance reading, which in our case is unknown. Regarding the landmark position sensors, the ALVAR tracking library does not provide any uncertainty measure for the estimated marker positions. Furthermore, the set of readings for these sensors contain outliers that can seriously affect the estimation. In this work we mitigate this fact by robustifying the error functions, replacing the least-squares quadratic cost function with the Huber one, which linearly weights the residuals that are greater than a certain threshold h.

To summarize, in our setup we identified four parameter that characterize the sensor uncertainty: a scale factor that multiplies the covariance matrix returned by the ICP algorithm, k_L, the Huber width for the displacement logical sensors, h_L, a diagonal 3D covariance matrix $\sigma_M I$, where I is the 3×3 identity matrix, associated to the fiducial maker position estimates, and the Huber width for the landmark position logical sensor, h_M. These parameters have to be tuned in order to achieve accurate calibration results.

To chose these parameters we can rely only on the output of the calibration process. For instance, once the robot poses and the camera misalignment have been estimated, it is possible to check whether the observed marker position with respect to the camera are consistent with the estimate of their absolute position $\mathbf{L}_i^{(W)}$, given the camera poses. In Figure 7.5 we have depicted an example of an *healthy* solution and one in which robot poses and/or the camera misaligned parameter most probably differ from the true values. In Figure 7.5a it is possible to see that the marker position measurements, which live in the camera frame S, once they have been moved to frame W by means of the estimated robot



(a) Healthy solution (b) Bad solution

Figure 7.5.: Absolute position estimates for one fiducial marker, with 99% confidence region (blue ellipse) and sensor sensor readings $z(t) = L^{(S)}(t)$, moved in frame W.

poses and camera misalignment, are consistent with the marker position estimate. Conversely this does not happen in Figure 7.5b. Other hints on the proper convergence of the solution might come from an analysis of the estimated robot trajectory smoothness: first recall that fiducial marker position measurements directly constraint robot poses, while odometry and scan matching readings yield velocity constraints that act on poses according to their integral over a moderately small time step. If the landmark position logical sensors are highly trusted, an outlier in their readings might result in large variations in subsequent robot poses, which is less likely to happen because of velocity outliers.

The calibration process was repeated with various assignment for the aforesaid tuning parameters, and the chosen ones were: $k_L = 2.5$, $H_L = 0.01$, $\sigma_M = 0.1$, $bH_M = 1$.

7.2. Reference Algorithm

To the best knowledge of the authors, no algorithm has been presented in the literature that can solve the considered, quite particular, pose tracking and parameter calibration problem. This does not mean that here we are dealing with a previously unsolved problem, yet we are arguing that there is no available algorithm or software framework that can solve that problem out-of-the-box or without the need of major extensions and/or adaptation, allowing us to discuss a full and fair comparison

of the results. Moreover, one of the key aspect of our solution is that it is general and flexible: for instance, if two camera were considered, or if the robot kinematic was omnidirectional insted of differential drive, we would just had to configure new logical sensors and/or extra calibration parameters in the problem setup. We argue that no other solution is available in the literature which possesses these features.

If a subset of the calibration parameters is considered, i.e., the wheel radius and the laser range-finders misplacement and misalignment, we can compare our calibration results with the ones obtained with the algorithm presented in [19], where a least-squares, closed form, solution is given for the simultaneous calibration of the wheel radii, robot baseline and single laser-range finder displacement and misalignment parameters. The code for the algorithm is freely available, so we can run the experiments on our data employing the original implementation, the default algorithm parameters and heuristics suggested by the authors. However, also for the reference algorithm, the laser range finder readings have to be pre-processed with a scan-matching algorithm to obtain relative 2-D transformations. In their algorithm, Censi et Al. employed their own implementation, which we replaced with the one employed on LURCH.

Another minor difference lies in the fact that the differential drive kinematic model considered in the reference algorithm is slightly different with respect to the one presented in Section 4.4.1, in the sense that a different left and right wheel radii are allowed. Moreover, since the reference calibration procedure can deal with a single range-finder only, we have to run it twice per each dataset, such that each run shares the odometry readings. The results for the left and the right runs are then averaged and compared with the ones obtained with ROAMFREE. Since the two problems considered are quite different, we provide the results as a reference only, without attempting to establish which of the two solutions is better.

7.3. Results

For each one of the seven datasets available we feed the ROAMFREE sensor fusion engine with the all collected readings and finally trigger the estimation, which is done by means of the Levenberg-Marquardt algorithm. In Table 7.1 we reported optimization problem statistics for each dataset employed.

We first discuss LURCH and fiducial marker position tracking results. In Figure 7.6 for each dataset we have plotted the LURCH absolute position error with respect to the ground truth, while in Table 7.2 we

DS	Nodes	Edges	LM iterations	Elapsed time
k1-1	3631	5478	41	4.84 s
k1-2	3640	5512	36	4.16 s
k1-3	3642	5533	37	4.07 s
k1-4	3645	5543	49	5.91 s
k1-5	3639	5543	40	5.02 s
k2-1	3641	5486	41	4.83 s
k2-2	3643	5509	42	4.64 s

Table 7.1.: Optimization problem statistics for the calibration runs.

have listed, for each dataset, the error of the estimated marker position. Moreover, we have marked with an exclamation mark the estimates that, given their uncertainty, lie outside a 95% confidence interval.

It is possible to see that the LURCH position is estimated with centimeter level accuracy in each dataset and that the estimation error is almost always lower than 5 cm. However, the marker position estimates are not always consistent with respect to the ground truth, while still showing remarkable accuracy. It is interesting to note that there exist markers, such as M12 and M13 for which the estimation error for each dataset is moderately high, although it is always below 20 cm, yet the average estimation error considering all the datasets is remarkably low, i.e., about 1 cm. Here an important role is played by the assumed uncertainty for the fiducial marker position readings, which we consider constant and equal to $\sigma_M I$. This might be an oversimplification. In fact recall that monocular camera images allow for angles only direct measurements and that marker positions can be determined on a metric scale because their dimension is known. This implies that the uncertainty associated to each marker reading has, at least, to increase with the marker distance. We argue that more consistent and, possibly, more accurate result could be obtained with a more sophisticated uncertainty model for the ALVAR tracking library results.

Next we evaluate the consistency of the LURCH pose estimates. In Figure 7.7 we have plotted, for each pose estimate and for each dataset, the Normalized Estimation Error Squared (NEES) [4], which let us compare the position uncertanitny with the real estimation error with respect to the ground truth. Here we assume that the estimation error at time t is unbiased and normally distributed with covariance matrix Σ_t, where Σ_t is the x-y pose uncertainty reported by ROAMFREE. Thus the squared estimation error is distributed as a χ^2_n distribution with n degrees of freedom, where $n = 2$ is the dimension of the estimate, since

we are considering the 2D position error only. We can compute 95% confidence interval for the NEES values according to [59]:

$$\left(1 - \frac{2}{9nM} - 1.96\sqrt{\frac{2}{9nM}}\right)^3 < e_t \Sigma_t^{-1} e_t^T < \left(1 - \frac{2}{9nM} + 1.96\sqrt{\frac{2}{9nM}}\right)^3, \tag{7.3}$$

where M is the number of estimates considered. In our case this the bound evaluate as $[0.013, 3.668]$. It is possible to see that NEES values for the great majority of the pose estimates, taken alone, lie within the considered bound. If we average the NEES values considering all the pose estimates, we can give a measure of the overall estimate consistency and give a tighter bound, obtaining the Average-NEES measure. Values above the higher bound suggest that the algorithm is optimist in reporting the estimate uncertainty, meaning that, on average, the estimation error is greater with respect the associated pose uncertainty. Conversely, conservative algorithms are characterized by ANEES values below the lower bound, which means that the estimation error is, on average, smaller if compared with the associated pose uncertainty. In this case we have $M = 20261$, the value for the Averaged NEES (ANEES) becomes 0.69, and the bound is $[0.986, 1.013]$, which gives a moderately conservative algorithm.

Next we discuss the calibration results and compare the obtained estimates with the ones from the reference algorithm by Censi et Al. See Table 7.3 and Table 7.4.

It is possible to see that the estimates for the wheel radius and the baseline are very stable and consistent with respect to direct inspection in each dataset. Indeed, the LURCH wheels are quite tick, so it is reasonable to assume few centimeters uncertainty in the baseline direct inspection. The reference algorithms retrieves analogous values, although it identifies a substantial difference between the left and the right wheel radius ($\approx$ 2.5 cm), which might be caused by different pressure in the wheel tires. Unfortunately, this is not handled in our kinematic model.

Regarding the laser displacements, both algorithm report values which are quite different with respect to the ones obtained by means of direct inspection. Moreover, our algorithm consistently reports high uncertainties for these parameters, as it happened in the previous Chapter with the GPS displacement parameter, implying that limited information can be obtained by other sensors regarding these quantities. Conversely, the laser misalignments are correctly retrieved by both algorithms. Note that while we provided symmetric initial guesses, the left laser true yaw is higher than 90°, and the right is lower than −90°, as it is has been obtained in calibration.

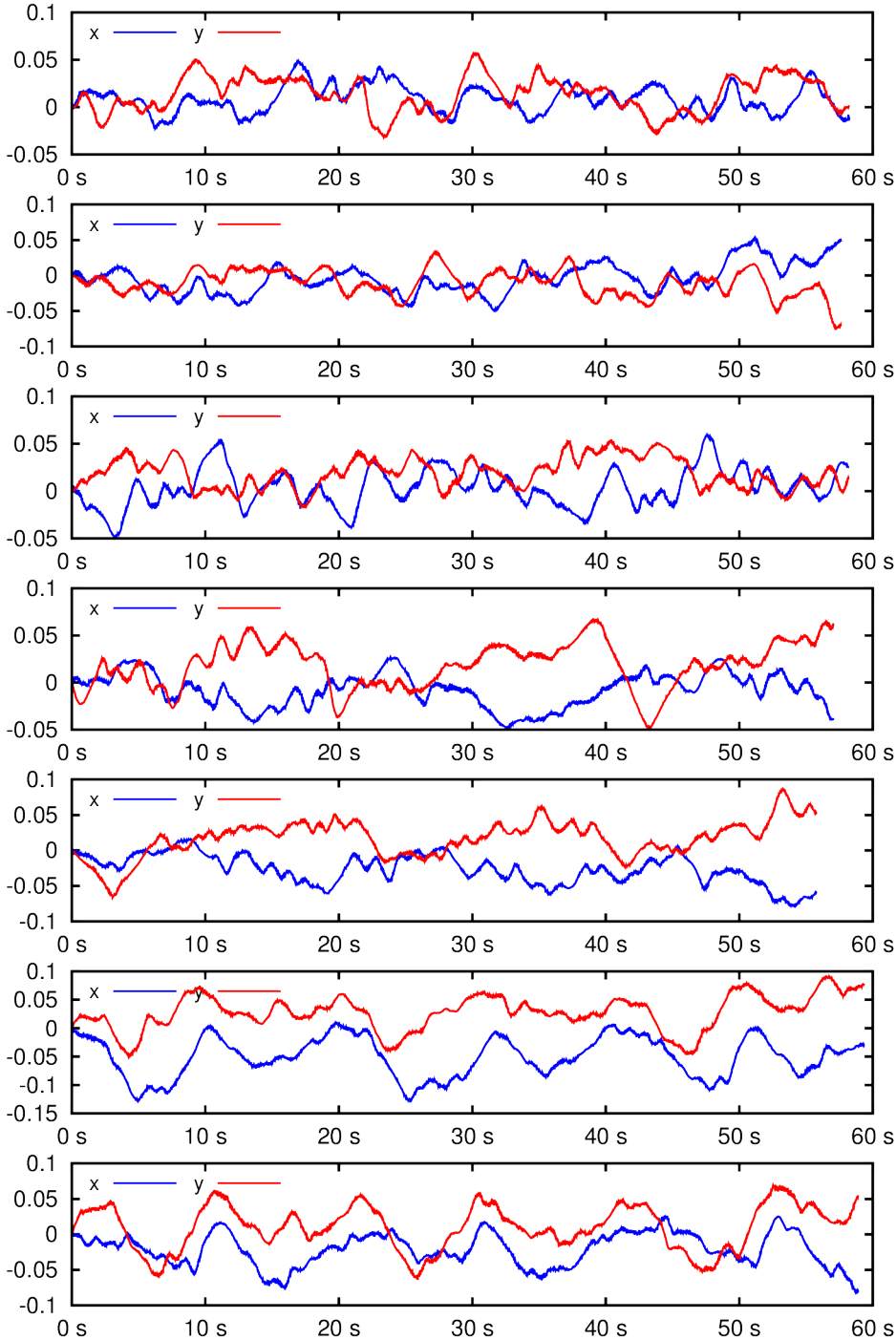


Figure 7.6.: LURCH 2D position error with respect to the ground truth for the calibration datasets.

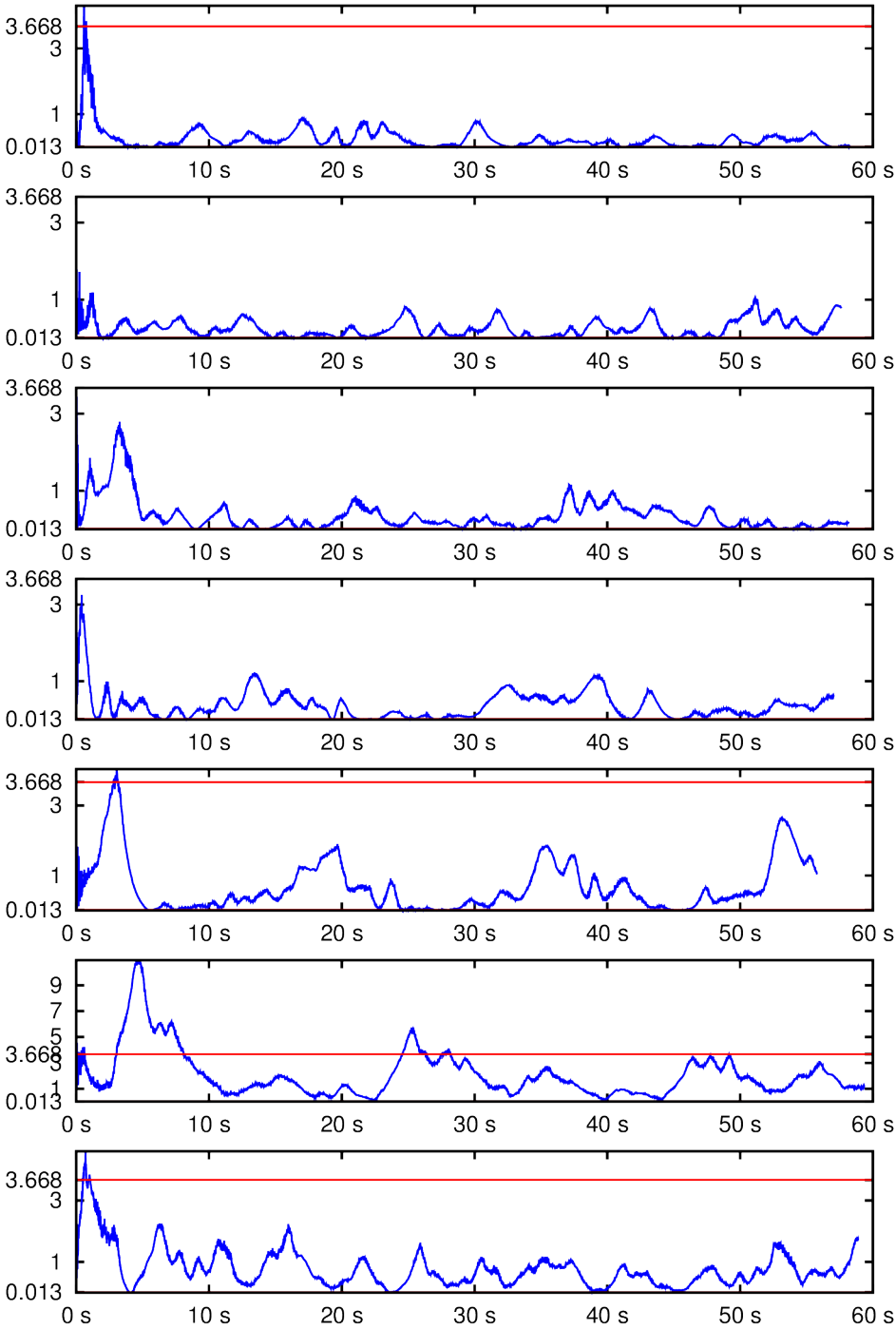


Figure 7.7.: LURCH NEES values for the calibration datasets.

		GT [m]	k1-1	k1-2	k1-3	k1-4	k1-5	k2-1	k2-2	avg
M1	x	−0.90	−0.09	−5.82 !	−2.68	0.13	−3.29	−8.93 !	−7.52 !	−4.03
	y	1.94	−1.65	5.08 !	6.39	−5.13	0.53	14.88 !	11.40 !	4.50
M2	x	−1.64	−9.66 !	7.54 !	1.71 !	−9.83 !	3.53 !	−10.36 !	−0.16	−2.46
	y	1.19	13.66 !	−6.50 !	−9.35 !	9.98 !	16.78 !	−7.18 !	−8.08	1.33
M3	x	−1.64	0.08	2.89	−2.22	−1.28	4.69	−12.76 !	−1.85	−1.49
	y	−0.00	7.67	−4.06	−3.05	5.54	7.10	−4.68 !	−7.96	0.08
M4	x	−0.00	4.90	−7.21	−6.10	3.81	0.01	−5.54 !	−8.60 !	−2.67
	y	1.93	0.19	1.89	3.71	−4.63	−2.60	15.82 !	13.17 !	3.94
M5	x	2.98	4.40	−11.68 !	−5.48 !	3.77 !	−8.61 !	1.96 !	−1.84 !	−2.50
	y	1.63	−9.67	14.27 !	14.42 !	−13.67 !	−7.82 !	12.52 !	11.52 !	3.08
M6	x	0.90	9.08 !	−9.93 !	−9.81 !	6.47	6.25	−3.66 !	−8.37 !	−1.42
	y	2.52	5.72 !	−2.39 !	−2.51 !	0.66	−2.97	13.33 !	13.24 !	3.58
M7	x	1.80	14.74 !	−15.83 !	−15.73 !	13.12 !	7.86 !	−8.75 !	−12.00 !	−2.37
	y	2.52	0.85 !	0.98 !	3.94 !	−1.21 !	−9.61 !	10.67 !	5.63 !	1.61
M8	x	3.40	−7.01	0.94	5.64	−9.60 !	−13.86 !	9.74	7.59	−0.94
	y	0.56	−5.63	12.09	9.01	−9.09 !	5.09 !	6.30	7.73	3.64
M9	x	2.86	−4.42	−0.28 !	2.37 !	−6.03 !	−9.97 !	10.02 !	3.46 !	−0.69
	y	−0.32	−6.30	12.62 !	9.90 !	−9.17 !	1.91 !	10.90 !	13.02 !	4.70
M10	x	2.29	−3.48 !	−1.49 !	3.81 !	−4.89 !	−14.40 !	14.16 !	8.74 !	0.35
	y	−1.26	−13.50 !	17.61 !	14.60 !	−14.84 !	−2.23 !	8.11 !	11.25 !	3.00
M11	x	−1.64	2.36	0.44	−3.60	0.94	5.09	−10.38 !	−1.57 !	−0.96
	y	−1.20	3.78	−2.41	0.41	3.70	4.20	−9.42 !	−9.91 !	−1.38
M12	x	−0.90	−15.02 !	11.83 !	12.91 !	−15.27 !	−12.30 !	7.57 !	14.86 !	0.65
	y	−1.94	−5.61 !	14.37 !	6.73 !	−5.56 !	12.27 !	−9.21 !	−3.95 !	1.29
M13	x	0.30	−10.39 !	9.69 !	8.66 !	−12.58 !	−8.91 !	6.99 !	11.15 !	0.66
	y	−1.92	−2.71 !	10.02 !	2.75 !	−1.67 !	10.86 !	−9.94 !	−4.57 !	0.68
M14	x	1.50	−5.68	6.85	5.77	−9.73 !	−7.20 !	9.92 !	10.09 !	1.43
	y	−1.87	−2.10	8.19	2.03	−2.43 !	7.36 !	−4.68 !	−0.82 !	1.08
avg	x		−1.44	−0.86	−0.34	−2.93	−3.65	−0.00	1.00	
	y		−1.09	5.84	4.21	−3.39	2.92	3.39	3.69	

Table 7.2.: Fiducial marker position estimation errors (in cm). The bold exclamation marks point out estimates which are not consistent with the GT, according to their uncertainty.

		guess	k1-1	k1-2	k1-3	k1-4	k1-5	k2-1	k2-2	avg
$\mathbf{r}$ [cm]		15.00	15.65 ±0.57	15.75 ±0.63	15.81 ±0.60	15.64 ±0.56	15.61 ±0.57	15.59 ±0.38	15.77 ±0.38	15.69 ±0.19
$\mathbf{L}$ [cm]		50.00	53.28 ±2.52	54.04 ±2.73	54.33 ±2.65	53.37 ±2.39	53.51 ±2.43	53.37 ±1.68	53.86 ±1.71	53.64 ±0.82
$\mathbf{S}_R^{(O)}$ [cm]	x	75.00	64.42 ±16.36	69.54 ±16.90	63.56 ±20.37	72.57 ±10.01	72.98 ±10.01	73.45 ±6.50	73.29 ±7.67	72.21 ±3.77
	y	−25.00	−17.72 ±13.70	−13.55 ±15.35	−14.06 ±14.14	−8.22 ±11.97	−13.02 ±13.19	−9.34 ±11.77	−0.99 ±12.19	−10.39 ±4.92
$\mathbf{R}_{S_R}^{O}$ [°]	θ	−90.00	−93.33 ±6.36	−88.35 ±6.49	−90.55 ±6.78	−84.01 ±5.29	−86.77 ±5.58	−77.96 ±4.84	−73.63 ±5.06	−83.39 ±2.13
$\mathbf{S}_L^{(O)}$ [cm]	x	75.00	57.82 ±8.24	53.58 ±8.45	56.91 ±8.20	62.02 ±7.40	56.26 ±8.55	72.70 ±5.16	75.55 ±4.63	66.39 ±2.50
	y	25.00	16.60 ±8.08	15.90 ±8.78	16.72 ±8.03	19.44 ±7.48	13.71 ±7.86	24.27 ±4.80	25.16 ±4.92	20.80 ±2.48
$\mathbf{R}_{S_L}^{O}$ [°]	θ	90.00	70.39 ±3.31	69.78 ±3.77	69.71 ±3.27	72.00 ±3.04	70.07 ±3.66	73.56 ±1.86	73.84 ±1.80	72.34 ±0.98
$\mathbf{R}_C^{O}$ [°]	w	0.00	0.36 ±0.05	0.36 ±0.04	0.36 ±0.04	0.36 ±0.04	0.36 ±0.04	0.37 ±0.03	0.37 ±0.03	0.37 ±0.01
	x	−0.50	−0.57 ±0.04	−0.57 ±0.04	−0.58 ±0.04	−0.58 ±0.04	−0.58 ±0.04	−0.58 ±0.02	−0.58 ±0.02	−0.58 ±0.01
	x	−0.50	−0.60 ±0.04	−0.60 ±0.04	−0.60 ±0.04	−0.59 ±0.04	−0.59 ±0.04	−0.59 ±0.02	−0.59 ±0.02	−0.59 ±0.01
	z	0.00	0.43 ±0.04	0.43 ±0.04	0.43 ±0.04	0.43 ±0.04	0.43 ±0.04	0.41 ±0.02	0.41 ±0.02	0.42 ±0.01

Table 7.3.: Calibration parameter results obtained with ROAMFREE.

		k1-1	k1-2	k1-3	k1-4	k1-5	k2-1	k2-2	avg
$\mathbf{r}$ [cm]	right	12.34	11.00	13.23	12.93	12.41	13.48	12.29	12.53
	left	15.34	15.19	14.64	14.71	15.66	14.61	15.89	15.15
	avg	13.84	13.10	13.94	13.82	14.04	14.05	14.09	13.84
$\mathbf{L}$ [cm]		57.32	54.15	49.52	56.86	59.21	49.82	61.57	55.49
$\mathbf{S}_R^{(O)}$ [cm]	x	73.21	73.61	69.26	72.00	73.23	70.38	73.74	72.20
	y	−30.32	−33.34	−23.36	−26.59	−26.93	−21.02	−27.95	−27.07
$\mathbf{R}_{S_R}^{O}$ [°]	θ	−86.22	−85.92	−87.53	−85.85	−85.78	−86.18	−85.46	−86.14
$\mathbf{S}_L^{(O)}$ [cm]	x	71.00	72.64	73.69	71.97	73.71	74.22	78.46	73.67
	y	45.37	45.84	29.30	42.29	46.69	29.38	57.13	42.29
$\mathbf{R}_{S_L}^{O}$ [°]	θ	72.12	71.45	71.49	72.63	72.38	71.53	72.62	72.03

Table 7.4.: Calibration parameter results obtained by means of the reference algorithm.

Finally, very similar value for the 3-D rotation from O to the camera frame C are obtained for each dataset. Even though it is difficult to intuitively visualize the transformation encoded by unit quaternions, it is possible to see that the retrieved ones substantially encode that the camera is pointed backwards and down, with an angle of $\approx 22°$. Unfortunately, we can not provide ground truth or other validation for these estimates. However, this parameter is critical for determining the absolute marker and robot positions, and we have shown that these quantities were determined very accurately in all the considered datasets.

In conclusion, we have shown how the calibration parameters of the LURCH Autonomous Wheelchair and the fiducial marker positions can be accurately retrieved by means of ROAMFREE, solving a simultaneous localization, mapping and calibration problem, similar, yet 6-DoF, to the one discussed in [53].

Chapter 8

An Integration Example

8.1. Introduction

In mobile robotic research, it is common to spend a considerable amount of time in developing, configuring and tuning hardware and software modules which are merely a precondition to develop higher level applications. These include, for instance, the electronics responsible of hardware driving and sensor reading, the communication infrastructure, the robot odometry and tracking software. Moreover, the ability of the robot to perform the desired higher level task often heavily depends on the quality of these modules [9]. While most of the cited problems could be considered solved in specific applications, only few solutions exist that are flexible enough to weakly depend on the platform they have been originally developed for. Moreover, robotics applications are often characterized by odd and/or stringent requirements over the properties or the performances of ground modules and the lack of off-the-shelf solutions often forces researchers to waste time in developing ad-hoc implementations as *premises* for their research.

While concepts such as reuse and abstraction are nowadays common in software engineering, it is still subject of active research how these can be transposed in the robotic field. Perhaps one of the most successful attempts is Robot Operating System (ROS) [69], which aims at supporting robot development and research by providing a modu-

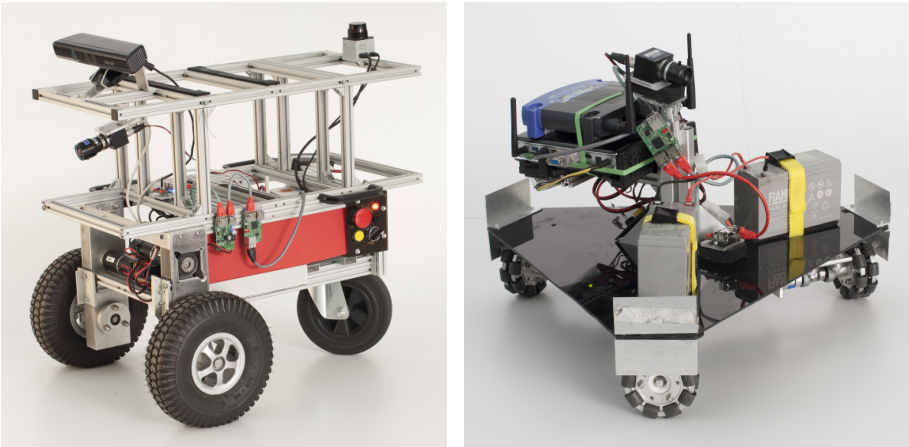
lar software framework, a communication infrastructure, and tools for debugging and inspecting robotic applications. Robot software implemented with ROS can benefit from a rich library of packages provided by other users, thanks to the open source development approach, significantly reducing the development time of a novel application. ROS is becoming a *de-facto* standard in robotics research software development; its very good reception and its widespread adoption have shown that the community needed a framework to join research efforts, boosting the development of new robotic applications. But is your next robot going to be supported in ROS? Although we believe that ROS did succeed in its goal, the vertical development of mobile robots, i.e., from mechanics to intelligence, still has many prerequisites that are often implemented from scratch, as application-dependent, non reusable solutions.

In this chapter we show how the ROAMFREE framework integrates with other open tools for the development of robotics applications. Use cases are described based on two mobile robots, Robocom [22], a heavy duty, differential drive robot and Triskar2, an omnidirectional robot designed for aggressive maneuvering in indoor environments (see Figure 8.1). The Rapid Robot Prototyping (R2P) framework [10, 11] is employed to implement the low level hardware and software layers, such as motor driving, odometry and perception, while ROS guarantees global communication and interfaces with other sensors such as cameras and laser range-finders. The resulting architecture shows how the basic hardware and software components needed by most of the robotic applications can be deployed in a modular and reusable way employing only off-the-shelf components. The performances of the proposed solution are evaluated by means of odometry benchmarks.

The next section reviews the R2P framework; in Section 8.3 we discuss the Robocom and Triskar2 architectures to show the advantages of reusable and modular hardware and software components. In Section 8.4 we validate the proposed approach by means of pose tracking benchmarks while in Section 8.5 we draw some conclusion and suggest future work.

8.2. Rapid Robot Prototyping (R2P)

A stepping stone for every mobile robot project concerns building and controlling the robot platform. A component based approach where robot platforms are assembled by connecting smart sensors and actuators in a commercial off-the-shelf fashion is foreseeable, but todays market supports very few devices focused on robotics applications. Robot



(a) Robocom (b) Triskar2

Figure 8.1.: The two target robots platform developed as a cases study.

designers are restricted to pick components from the hobby market (e.g., Arduino[1]), which are cheap, but their performances are not suitable for complex systems, or from the industrial automation field, however those are expensive and their size, weight, and power requirements are often not suitable for long endurance battery powered operations. The common alternative is to design custom devices, although this requires specific skills and could significantly slow down the development process. Another common issue lies in the need to connect these devices together, eventually with real-time constrains, and interface them with a computer for high-level tasks.

To simplify the development of new robotic platforms, the Rapid Robot Prototyping (*R2P*) [10, 11] was proposed, an open-source hardware and software framework aimed at speeding up the prototyping of robotic systems. R2P relies on the principle that the requirements of a generic robotic platform can be implemented by modules not only at software level, as it is common in most robotics frameworks, but also at hardware level. Modular platforms for robotics have been proposed in recent years [39, 64], allowing to easily build robots for research in specific areas (e.g., swarm robotics); with R2P, we extend the modular development approach to a wide range of platforms, from small mobile robots to autonomous outdoor veichles, which require performance and generality not achievable with domain-specific architectures.

In R2P, basic functionalities such as motor control, trajectory follow-

[1] http://www.arduino.cc

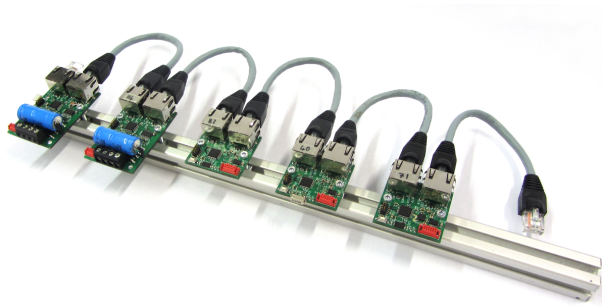


Figure 8.2.: A set of R2P modules connected together.

ing, inertial data acquisition, have been implemented by specific, standardized, hardware modules, with corresponding firmware, that can be plugged on a common bus and interact in real-time to compose a robotic application. Each R2P module is focused on a particular functional requirement, providing the electronics needed to perform a specific task (e.g., the power stage to drive a motor), and a microcontroller running embedded software to control the hardware (e.g., a PID algorithm to follow a speed setpoint). Modules are connected using a single cable providing both power supply and a CAN bus for data exchange. A daisy-chain connection schema is exploited, reducing wires and allowing for easy addition of components to existing systems. Real-time communication is provided by RTCAN [62], a CAN bus protocol that supports sporadic, event-triggered, and periodic, time-triggered, communication according to hard and soft real-time constrains. Furthermore, it ensures global clock synchronization on the real-time R2P network within a margin of few microseconds, so that data, e.g., sensor readings or event signals, can be precisely timestamped.

To support users in writing the code running on the hardware modules, R2P includes a lightweight middleware, following the publish/subscribe model. Computation is performed by a network of nodes, which interact by declaring topics and publishing and/or subscribing messages based on those topics. This loosely-coupled communication paradigm allows the writing of reusable software components at the firmware level, as in other common robotics frameworks targeted at desktop-class computers (e.g., ROS, OROCOS, etc.): low-level functionalities can be implemented by developing new nodes, which can then be shared by different projects.

At the moment of writing, R2P features a DC motor controller, a brushless motor controller, a MEMS inertial measurement unit featuring accelerometer, gyroscope, compass, pressure sensor, and a GPS input, a range finder module supporting infrared and ultrasonic sensors, and a

DC-DC power supply module. Additionally, a gateway module provides USB and Ethernet connectivity, allowing R2P modules to communicate with a computer. Integration with ROS is provided by µROSnode [63], a lightweight, open source, ANSI C ROS client library. Data published on the R2P network can be accessed from ROS nodes through the Ethernet connection, and, in the same way, R2P modules can subscribe to topics published by ROS nodes, without the need of specific drivers to interface the hardware platform with software developed in ROS.

In addition to the hardware modules, several ready-to-use nodes come with R2P, providing useful functionalities like filtering algorithms (e.g., to estimate pose and attitude from inertial sensor readings), control algorithms (e.g., PID controller), forward and inverse kinematics models (e.g., differential, Ackermann, and omnidirectional kinematics), and others, to drastically reduce the effort for the development of embedded firmware in common mobile robot architectures.

Exploiting R2P modules, complex robotic platforms can be built in a plug-and-play fashion by simply selecting off-the-shelf R2P modules according to application functional requirements and by defining their interactions with an easy to use programming environment. Integration with ROS allows users to develop complex applications, while low-level control is implemented by means of a modular distributed architecture, with real-time performance, without the need for advanced domain-specific knowledge. R2P is an open source project[2]; schematics and layouts of the hardware modules are released under CC-BY-SA license, while the firmware code, the RTCAN protocol, the middleware, and µROSnode are distributed with a BSD 2-Clause license.

8.3. A Case Study in Mobile Robots Development

In this section we discuss a case study based on two mobile robots, Robocom and Triskar2 (see Figure 8.1), and we show how R2P and ROAMFREE have been employed to implement two of the most basic capabilities of an autonomous mobile platform: being able to move and perceive its own movement. The resulting hardware and software architecture is highly general and, thanks to the modularity of the tools employed, it can be easily deployed in other indoor and outdoor scenarios (e.g., the development of an unmanned aerial vehicle).

The proposed architecture is built on two essential components: an ecosystem of R2P modules, implementing the low-level, real-time, communication infrastructure, running control loops, and providing times-

[2] http://github.com/openrobots-dev

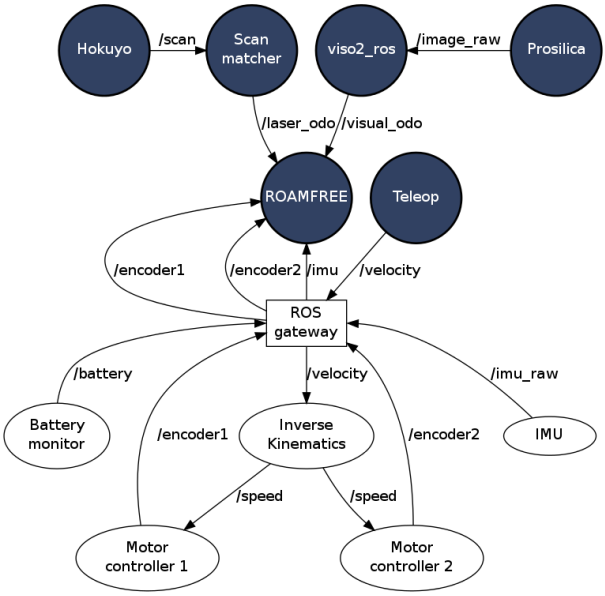


Figure 8.3.: The software architecture of Robocom. White ellipses represent embedded R2P nodes, while blue circles are ROS nodes.

tamped sensor readings, and a standard computer running ROS and hosting the ROAMFREE pose tracking node. Here ROS provides the *glue* between the hardware and software domain, as it allows to plug into the architecture other off-the-shelf hardware and software modules available in ROS. Figure 8.3 shows the Robocom software architecture.

The low-level control architecture of the two robots is based on R2P DC Motor modules (two on Robocom, three on Triskar2), a R2P Inertial Measurement Unit module, a R2P Power Supply and the R2P Gateway which is connected to an onboard computer through the Ethernet connection. Figure 8.4 shows the architecture for the Triskar2 robot. R2P DC motor modules run closed loop controllers to drive the wheels and publish encoder readings. The IMU provides acceleration, angular velocity and Earth magnetic filed readings at 100 Hz. Robots are teleoperated by means of a remote ROS node publishing setpoints in terms of linear and angular velocity. The Gateway module converts messages from standard *TCPROS* topics to the real-time R2P domain, and the other way around. One of the R2P DC motor modules hosts a R2P node which subscribes to the velocity setpoint topic, computes the inverse kinematics and publishes the obtained wheel speed setpoints for the motor controllers. The wheel speed control loops run on the motor

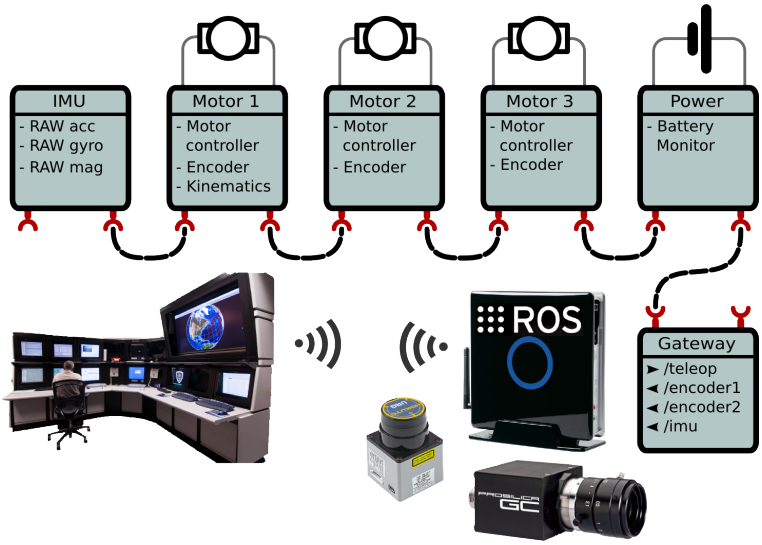


Figure 8.4.: The hardware architecture of Triskar2, with the daisy-chained R2P modules, the onboard computer running ROS and the sensors employed. A wireless link is used to remotely operate the robot.

controllers, with an update frequency of 100 Hz.

ROAMFREE exists as a *rospy* node on the onboard computer which is connected to the R2P network by means of the R2P Gateway module. It subscribes to sensor readings (either from R2P modules or from other ROS nodes on the local host), configures logical sensors, drives the pose tracking loop and publishes estimates as a *tf* transformation. In the Listing 8.1 we show how a differential drive kinematics logical sensor can be configured by means of the ROAMFREE Python API to handle the Robocom encoder readings. In the last two lines the kinematics constants, i.e., the wheel radius and distance, are configured as constant parameters: note the use of the `fixed` argument: it controls if the parameter has to be tracked during estimation, in which case the value provided would be employed just as an initial guess.

After configuring logical sensors, the ROAMFREE node subscribes to the topics providing sensor readings. Employing to the ROS publish/subscribe mechanism a callback function is set to be invoked when new messages are available. In Listing 8.2 we show an example of such a method which feeds gyroscope readings into the ROAMFREE sensor fusion engine. Note that this code allows to treat any kind of angular velocity measurement source, such as visual odometry or scan-matching algorithms, although minor changes may be needed to extract the actual

```
RF.addSensor('Odo', master=True,
   type=SensorTypes.DiffDriveKinematic)

RF.setSensorDisplacement('Odo', [0.0, 0.0, 0.0], fixed=True)
RF.setSensorMisalignment('Odo', [0.0, 0.0, 0.0], fixed=True)

RF.addParameter(ParameterTypes.Euclidean1D,
   'Odo_R', [0.12], fixed=True) # wheel radius
RF.addParameter(ParameterTypes.Euclidean1D,
   'Odo_L', [0.45], fixed=True) # ... and distance
```

Listing 8.1: Configuring the differential drive kinematics logical sensor for the Robocom robot. Here `Odo` is the architecture master sensor, i.e., the one which triggers pose estimates to be instantiated when new sensor reading are available.

sensor reading from the ROS message.

From the user point of view, all the internals of the complex formulation of the sensor-fusion problem are encapsulated in the *addMeasurement* method. In this case its parameters tell the Graph Management component (see Section 3.5) that a new reading is available for the gyroscope sensor, which has been previously configured in a similar way with respect to Listing 8.1. Since the master sensor is the differential drive kinematics one, this call does not trigger new pose nodes to be added to the factor graph. Instead, an angular velocity constraints is inserted so that it is incident to the nearest pose with respect to the measurement timestamp, the previous one employed for the gyroscope sensor, and all the involved logical sensor calibration parameters.

In our case study, beside the R2P IMU and wheel encoders, we extend the set of sensors available for odometry estimation with a Prosilica GC750C camera, providing greyscale 752x480 timestamped images and a Hokuyo URG-04LX laser range-finder sensor with a 240° field of view. Images from the camera feed the ROS wrapper for the *libviso2* monocular visual odometry software package [33]. This library provides velocity estimates on a metric scale assuming known and fixed the height and the pitch of the camera with respect to the ground plane. The output is employed to construct a ROAMFREE linear and angular speed logical sensor. Another logical sensor is obtained by the output of a 2-D scan-matcher node implementing an Iterative Closest/Corresponding Point (ICP) algorithm, with accurate covariance estimation [17], which employs the Hokuyo scan data. All the software employed is already available in ROS.

The importance of precise timestamping of sensor readings has been

```
def gyro_callback(msg):
  z = [ msg.angular_velocity.x,
        msg.angular_velocity.y,
        msg.angular_velocity.z]
  T = timeFromROSheader(msg)

  RF.addMeasurement('Gyro', T, z, z_cov)

rospy.Subscriber("/R2Pimu", sensor_msgs/Imu, gyro_callback)
```

Listing 8.2: Defining a callback function to handle gyroscope readings pubblished as standard ROS messages.

often pointed out in the literature, see for instance [40]. In the proposed architecture, the timestamps are read from the header of ROS messages, which are not the ones set by *roscore* as messages enter the publish/subscribe network. These values are filled by the publisher nodes and they are set to be as close as possible to the time in which the physical phenomena actually took place. In the proposed architecture, this is guaranteed for most of the sensors: (i) R2P modules share a precise distributed clock which is employed to timestamp messages in the R2P network, (ii) as it is being done by other sensors, such as Prosilica cameras, the Precision Time Protocol (PTP) is employed by the R2P gateway to synchronize its clock with the onboard computer one. Thanks to the ROAMFREE out-of-order, delayed measurements handling and the precise sensor timestamping, it is possible to accommodate for network and processing latencies.

8.4. Experimental Results

In this section we discuss odometry benchmarks performed on Robocom and Triskar2. In these experiments we stress the sensors, the communication infrastructure, and the pose tracking system to show that the resulting hardware and software architecture is ready to be employed in research in most of the common higher level application areas. It is important to note that it is beyond the scope of the current discussion to show that the presented frameworks perform better than any other given solution. Conversely, we want to emphasize the fact that, thanks to the flexibility and the modularity of R2P and ROAMFREE, it is easy to deploy an effective and reusable robot architecture at a fraction of the development cost usually required by an ad-hoc solution (e.g., a few days including wiring).

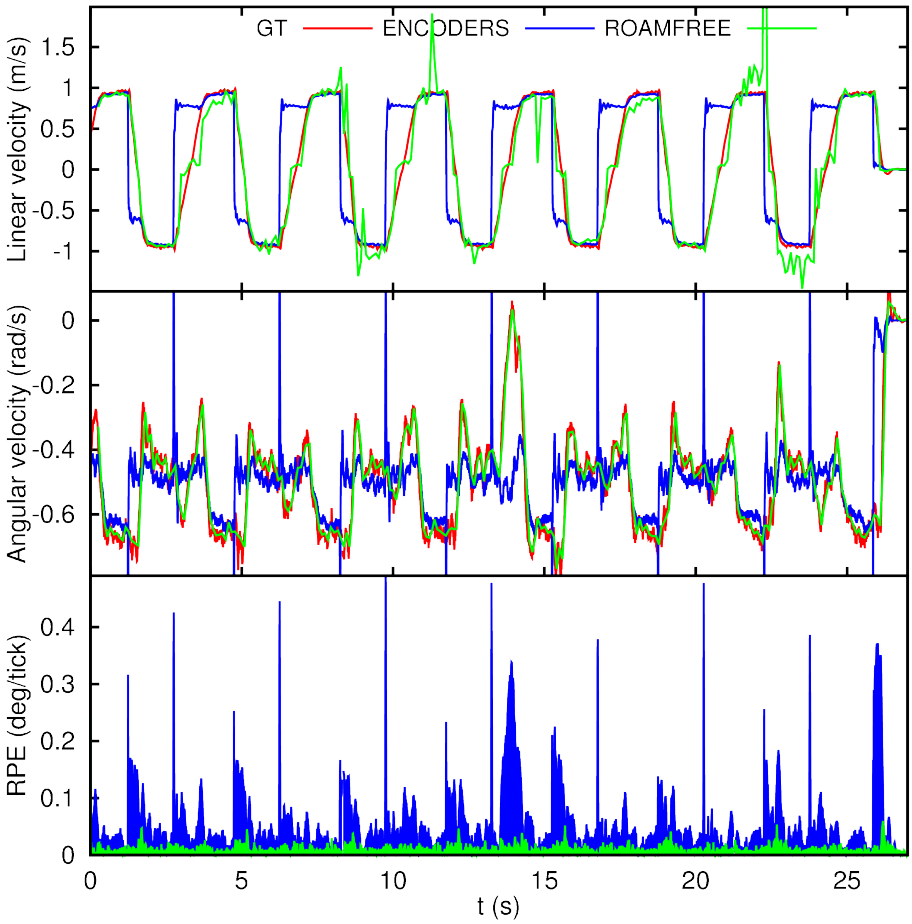


Figure 8.5.: Robocom odometry experiments showing ground truth, direct kinematics and ROAMFREE estimates for the linear and angular velocity, along with rotational RPE.

We first address the problem of calibrating sensor parameters and kinematics constants, which is often addressed by means of direct measurements on the robots. However, this approach can be impractical or even impossible in some situations. Moreover, more accurate values can often be obtained comparing direct kinematics estimates with an external ground truth. In this work we exploit the multi-sensor self-calibration capabilities of ROAMFREE to obtain estimates for the required parameters without the need of an external ground truth. In our case, these consist in the kinematics constants r, the wheel radius, and L, which is the axis length for the differential drive kinematics and the

distance from each wheel to the center of mass in the omnidirectional case. Moreover, on Triskar2, the Hokuyo laser rangefinder is rotated and translated with respect to the robot reference frame to exploit the whole sensor field of view. Accurate values for this 2D roto-translation, Δx, Δy, and θ, are required to correctly refer local linear velocity readings to the robot frame. We use direct measurements of these parameters as initial guesses for an offline parameter calibration step employing ROAMFREE. We start from *rosbag* datasets containing visual odometry, scan-matching, encoders and inertial sensors readings collected during random robot movements along smooth and slow trajectories, such as to prevent wheel slippage. As in Listing 8.1 we configure ROAMFREE logical sensors and we unfix the unknown parameters. Then we ask for the solution of the pose tracking and parameter calibration problem considering the whole, ~ 60 s, datasets. The results were $r = 124$ mm, $L = 461$ mm for Robocom, while for Triskar2 we have $r = 67$ mm, $L = 337$ mm, $\Delta x = 153$ mm , $\Delta y = -288$ mm and $\theta = -54.85°$. These values are slightly different from the initial guesses, yet improving the likelihood of the tracked poses given the whole set of sensor readings. These calibration parameters are employed in the later pose tracking experiments.

We next benchmark the accuracy of the pose tracking driving the robots by means of high amplitude square waves in the tangential and angular velocity setpoints, so that relevant wheel slippage occurs. In this situation encoders and direct kinematics alone are unable to provide accurate odometry estimate. Nevertheless, thanks to ROAMFREE the available scan-matching, visual odometry and inertial sensor readings can be employed to improve the odometry estimate. It is important to note that the robot pose estimate will eventually diverge from the ground truth since none of the available sensors is employed to estimate the absolute robot position and/or orientation with respect to a map, which could in principle be done for example employing the Prosilica camera and a monocular SLAM algorithm or some global localization algorithm based on laser readings (e.g., gmapping). We compare the estimated linear and angular velocities with respect to the ground truth and we evaluate the rotational Relative Pose Error [14]: for each couple of poses we compare the relative rotation between them with the corresponding ground truth values. In Figure 8.5 we present the results for the Robocom robot. This robot mounts two 150 W Maxon motors which are able to suddenly reverse the wheel speed, incurring in major slippage. Thus, if only the encoder readings were considered, substantial error would be introduced in odometry estimate, as it is possible to see in Figure 8.5, for instance, at $t = 2.5$ s, where the direct kinematics

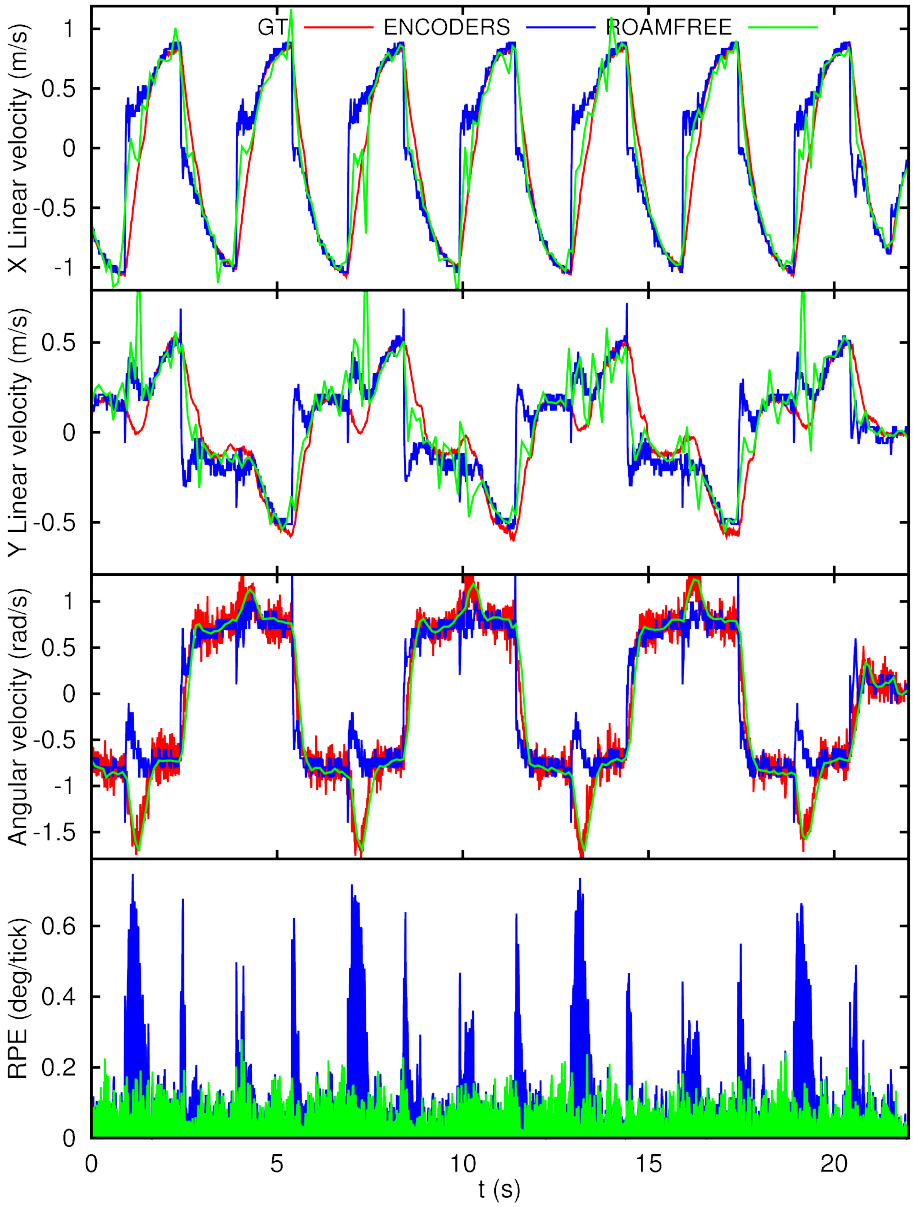


Figure 8.6.: Triskar2 odometry experiments showing ground truth, direct kinematics and ROAMFREE estimates for the linear and angular velocity, along with rotational RPE.

estimate (blue line) substantially differs from the ground truth (red line) as the tangential speed setpoint is reversed. Conversely, ROAMFREE is able to correct for most of the error on the linear velocity and to track

the angular velocity very accurately, as it can be seen from the RPE.

Next we consider similar experiments for the Triskar2 robot. Again we drive the robot in a way such that odometry estimation by means of direct kinematics becomes unreliable because of slippage. In this experiment, a further issue consists in the significant vibrations introduced by the omnidirectional wheel rollers. The resulting noise affects gyroscope and accelerometer measures and blurs the Prosilica images, introducing outliers in the inertial and visual odometry sensors. Although an advanced outlier rejection mechanism is under development in ROAMFREE, at the present stage of development one has to rely only on error function robustification. Still, it is possible to see in Figure 8.6 that most of the slippage is compensated employing other sensor readings.

8.5. Conclusions

In this chapter we have reviewed ROAMFREE and R2P frameworks and we have shown how the core architecture for a mobile robot, sporting motion, control, multi-sensor fusion, and self-calibration capabilities can be built using only off-the-shelf hardware and software components. The resulting architecture features a high degree of generality and, thanks to the flexibility and modularity of the employed frameworks, can be easily deployed on very different mobile robot platforms. Moreover, odometry benchmarks have been discussed in which motor control, communication infrastructures and sensor-fusion software have been stressed. The result shows that, thanks to the precise timestamping of sensor readings and state-of-the-art factor graph based sensor fusion techniques, both Robocom and Triskar2 are able to accurately and robustly estimate their own motion, a feature which is a prerequisite for most of high-level robotic applications. The deployment of the presented hardware and software frameworks, which are available as open hardware and open software components, can be easily done at a fraction of the cost of the development of an ad-hoc solution, eventually enabling to boost research in advanced applications.

Chapter 9

Discussion

In this thesis we have presented ROAMFREE, a generic and modular framework for multiple sensors information fusion and parameter self-calibration, with application in mobile robotics. In the following we discuss some open issues and propose future improvements.

As in any generic framework, modularity and flexibility come at a cost. The development of such a system required the design of a complex software architecture achieving the decoupling between sensor hardware, state variable representations, measurement error models and solver algorithms, whose structure is presented in this work.

We do not see in the complexity of the resulting code any real cost for the end user since we provided easy to use interfaces to handle it; the real cost in ROAMFREE is related to the key assumption behind logical sensors. On the one hand logical sensors provide a fundamental abstraction which makes the system independent with respect to the actual platform and hardware sensors employed. On the other hand, the *black-box* assumption prevents solvers to deal with the internals of the actual sensor or processing algorithm. To understand the impact of this, consider for instance a visual odometry system that tracks simultaneously the camera egomotion and a set of world-fixed features. In our approach, such a system is handled as a black-box linear and angular velocity information source, possibly characterized by calibration parameters such as a scale factor in the monocular case. This approach ultimately hides the fact

that camera and landmark poses have to be maintained internally by the visual odometry algorithm, as they are decoupled from the ROAMFREE internal representation. The instantaneous linear and angular velocity output of the visual odometry system is employed by ROAMFREE, possibly along with other information sources such as a GPS or an IMU, to determine final pose estimates by means of sensor-fusion. Since no feedback is provided from the framework to the visual odometry algorithm, the pose estimate internally maintained by the latter will eventually diverge from the one computed by the framework, resulting in a possible source of inconsistency

It is subject of ongoing research to determine how the ROAMFREE pose estimate could be employed to provide general feedback loops to logical sensors. Regarding the example above, the internal state of the visual odometry algorithm, i.e., the position of the features and the camera, could be corrected taking into account the estimate obtained with the full set of sensors. This would further increase the overall complexity of the framework; hiding this complexity to the end user, which could also be the actual information source developer, is the next challenge.

Another critical point regards parameters self-calibration. We have provided a formulation for the sensor-fusion problem that, in principle, should allow to solve the simultaneous localization and parameter self-calibration problem. Even though the experimental evaluation carried out so far shows that this formulation does work (in the considered cases), there exist examples of calibration parameters, such as the sensor displacements, that are not retrieved correctly or with higher confidence intervals with respect to direct inspection. In other cases, we faced limitations because of 2-D motion. Furthermore, the estimated values in certain situations heavily depend on their initial guess, which can also prevent proper convergence of the final pose estimates in calibration runs. This happened with the steering gain coefficient in Ackermann kinematics. In that case we were forced to develop a specific calibration heuristic.

Because of these issues, while we argue that we succeeded in delivering out-of-the-box online pose tracking, as we demonstrated in the experimental part of this work, regarding parameter self calibration this result is yet to come. At the present stage of development, we are not able to give general results on which conditions have to hold for a given parameter to be determined. A rigorous observability analysis is needed, allowing the framework to automatically detect which parameters can be determined given the available information sources and the trajectory of the robot.

As a direction for future improvements, we also point out another issue related to measurement timestamps. At the moment, ROAMFREE assumes that all the information sources are synchronized and thus the provided timestamps can be safely employed to determine the relative order of sensor readings and to associate them to the correct robot poses. However, this assumption is seldom satisfied in practice. We have seen an example of this on the Quadrivio ATV, where, because of communication latencies, the GPS timestamps, associated to sensor readings at the on-board computer, were delayed in a non-negligible way.

Although the synchronization of information sources could be addressed by carefully designing the communication infrastructure, e.g., by employing real time protocols and clock distribution algorithms as we have done on the Robocom and Triskar2 robots, it would still be desirable to be able to handle latencies and inaccurate timestamps in the sensor-fusion framework itself. Also note that there are sensor-fusion problems which are sensitive to millisecond-scale timing latencies, as the one faced in Zebedee, a spring-mounted, laser range-finder coupled with an IMU [12] for 3-D mapping applications. One way to handle these issues would be to augment the optimization state with extra dimensions to estimate latency corrections over the fixed-lag window.

As a final remark, we note that the developed sensor fusion library it is already able to solve the *full* SLAM problem. Although an extensive evaluation should be performed to support this claim, the calibration for the LURCH autonomous wheelchair was performed by simultaneously determining the pose of the robot and the map of the environment, in that case consisting in the fiducial markers positions. More complex cases could be considered in which the fiducial markers were replaced with scale invariant features detected in camera images, as it happens in visual SLAM algorithms, or with laser scans associated with robot poses, which is currently an ongoing work.

Bibliography

[1] M. Abe and W. Manning. *Vehicle handling dynamics: theory and application*. Butterworth-Heinemann, 2009.

[2] Pratik Agarwal, Gian Diego Tipaldi, Luciano Spinello, Cyrill Stachniss, and Wolfram Burgard. Robust map optimization using dynamic covariance scaling. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 62–69. IEEE, 2013.

[3] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[4] Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. Wiley-Interscience, 2001.

[5] Gianluca Bardaro, Davide Antonio Cucci, Luca Bascetta, and Matteo Matteucci. A simulation based architecture for the development of an autonomous all terrain vehicle. In *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 74–85. Springer International Publishing, 2014.

[6] L. Bascetta, G.A. Magnani, P. Rocco, and A.M. Zanchettin. Design and implementation of the low-level control system of an all-terrain mobile robot. In *Advanced Robotics (ICAR), 2009 International Conference on*, pages 1–6. IEEE, 2009.

[7] Luca Bascetta, Davide Cucci, Gianantonio Magnani, Matteo Matteucci, Dinko Osmankovic, and Adnan Tahirovic. Towards the implementation of a mpc-based planner on an autonomous all-terrain vehicle. In *In Proceedings of Workshop on Robot Motion Planning: Online, Reactive, and in Real-time (IEEE/RJS IROS 2012)*, pages 1–7, 2012.

[8] Rossella Blatt, Simone Ceriani, Bernardo Dal Seno, Giulio Fontana, Matteo Matteucci, and Davide Migliore. Brain control of a smart wheelchair. In *10th International Conference on Intelligent Autonomous Systems*, 2008.

[9] Andrea Bonarini. The body, the mind or the eye, first? In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *Lecture Notes in Computer Science*, pages 210–221. Springer Berlin Heidelberg, 2000.

[10] Andrea Bonarini, Matteo Matteucci, Martino Migliavacca, and Davide Rizzi. R2P: an open source modular architecture for rapid prototyping of robotics applications. In *Proceedings of 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT'12)*, 2012.

[11] Andrea Bonarini, Matteo Matteucci, Martino Migliavacca, and Davide Rizzi. R2P: An open source hardware and software modular approach to robot prototyping. *Robotics and Autonomous Systems*, 2013.

[12] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *Robotics, IEEE Transactions on*, 28(5):1104–1119, 2012.

[13] Denis Bouvet and Gaetan Garcia. Improving the accuracy of dynamic localization systems using rtk gps by identifying the gps latency. In *Robotics and Automation (ICRA), 2000 IEEE International Conference on*, pages 2525–2530. IEEE, 2000.

[14] Wolfram Burgard, Cyrill Stachniss, Giorgio Grisetti, Bastian Steder, Rainer Kummerle, Christian Dornhege, Michael Ruhnke, Alexander Kleiner, and Juan D Tardós. A comparison of slam algorithms based on a graph of relations. In *Intelligent Robots and Systems (IROS), 2009 IEEE/RSJ International Conference on*, pages 2089–2095. IEEE, 2009.

[15] Luca Carlone, Andrea Censi, and Frank Dellaert. Selecting good measurements via ℓ_1 relaxation: a convex approach for robust estimation over graphs. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, page to appear. IEEE, 2014.

[16] Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman, and Frank Dellaert. Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors. *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[17] Andrea Censi. An accurate closed-form estimate of icp's covariance. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3167–3172. IEEE, 2007.

[18] Andrea Censi, Antonio Franchi, Luca Marchionni, and Giuseppe Oriolo. Simultaneous calibration of odometry and sensor parameters for mobile robots. *Robotics, IEEE Transactions on*, 29(2):475–492, 2013.

[19] Andrea Censi, Luca Marchionni, and Giuseppe Oriolo. Simultaneous maximum-likelihood calibration of odometry and sensor parameters. In *Robotics and Automation (ICRA), 2008 IEEE International Conference on*, pages 2098–2103. IEEE, 2008.

[20] Andrea Censi and Davide Scaramuzza. Calibration by correlation using metric embedding from nonmetric similarities. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(10):2357–2370, 2013.

[21] Simone Ceriani. *Conditionally independent visual slam with integrated bundle adjustment*. PhD thesis, Italy, 2013.

[22] Simone Ceriani, Giulio Fontana, Alessandro Giusti, Daniele Marzorati, Matteo Matteucci, Davide Migliore, Davide Rizzi, Domenico G Sorrenti, and Pierluigi Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27(4):353–371, 2009.

[23] John Cocke. Global common subexpression elimination. *ACM Sigplan Notices*, 5(7):20–24, 1970.

[24] Davide Antonio Cucci and Matteo Matteucci. A flexible framework for mobile robot pose estimation and multi-sensor self-calibration. In *Informatics in Control, Automation and Robotics (ICINCO), 2013 International Conference on*, pages 361–368, 2013.

[25] Davide Antonio Cucci and Matteo Matteucci. On the development of a generic multi-sensor fusion framework for robust odometry estimation. *Journal of Software Engineering for Robotics*, 5(1):48–62, 2014.

[26] Davide Antonio Cucci and Matteo Matteucci. Position tracking and sensors self-calibration in autonomous mobile robots by gauss-newton optimization. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1269–1275. IEEE, 2014.

[27] Davide Antonio Cucci, Martino Migliavacca, Andrea Bonarini, and Matteo Matteucci. Development of mobile robots using off-the-shelf open-source hardware and software components for motion and pose tracking. In *Intelligent Autonomous Systems (IAS), 2014 International Conference on*, page to appear, 2014.

[28] DARPA. Adaptable navigation systems. http://www.darpa.mil/Our_Work/STO/Programs/Adaptable_Navigation_Systems_(ANS).aspx.

[29] Timothy A Davis. *Direct methods for sparse linear systems*, volume 2. Siam, 2006.

[30] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. 2012.

[31] Mark Fiala. Artag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 590–596. IEEE, 2005.

[32] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[33] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[34] Philip E Gill and Walter Murray. Algorithms for the solution of the nonlinear least-squares problem. *SIAM Journal on Numerical Analysis*, 15(5):977–992, 1978.

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

[36] Mohinder S Grewal and Angus P Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]. *Control Systems, IEEE*, 30(3):69–78, 2010.

[37] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010.

[38] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2003.

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

[40] Alastair Harrison and Paul Newman. Ticsync: Knowing when things happened. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 356–363. IEEE, 2011.

[41] Mengwen He, Huijing Zhao, Jinshi Cui, and Hongbin Zha. Calibration method for multiple 2d lidars system. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, page to appear. IEEE, 2014.

[42] Tom Henderson and Esther Shilcrat. Logical sensor systems. *Journal of Robotic Systems*, 1(2):169–193, 1984.

[43] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz SchröDer. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77, 2013.

[44] Steve Hewitson, Hung Kyu Lee, and Jinling Wang. Localizability analysis for gps/galileo receiver autonomous integrity monitoring. *Journal of Navigation*, 57(2):245–259, 2004.

[45] Radu Horaud and Fadi Dornaika. Hand-eye calibration. *The international journal of robotics research*, 14(3):195–210, 1995.

[46] Peter J Huber. *Robust statistics*. Springer, 2011.

[47] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. Factor graph based incremental smoothing in inertial navigation systems. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 2154–2161. IEEE, 2012.

[48] Simon J Julier and Jeffrey K Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. symp. aerospace/defense sensing, simul. and controls*, volume 3, pages 3–2. Orlando, FL, 1997.

[49] M. Kaess, S. Williams, V. Indelman, R. Roberts, J.J. Leonard, and F. Dellaert. Concurrent filtering and smoothing. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 1300–1307, 2012.

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

[51] TR Kronhamn. Bearings-only target motion analysis based on a multihypothesis kalman filter and adaptive ownship motion control. *IEE Proceedings-Radar, Sonar and Navigation*, 145(4):247–252, 1998.

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

[53] R. Kümmerle, G. Grisetti, and W. Burgard. Simultaneous parameter calibration, localization, and mapping. *Advanced Robotics*, 26(17):2021–2041, 2012.

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

[55] R Kummerle, Giorgio Grisetti, and Wolfram Burgard. Simultaneous calibration, localization, and mapping. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3716–3721. IEEE, 2011.

[56] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee, 1991.

[57] John J Leonard, Paul M Newman, Richard J Rikoski, José Neira, and Juan D Tardós. Towards robust data association and feature modeling for concurrent mapping and localization. In *Robotics Research*, pages 7–20. Springer, 2003.

[58] Hongdong Li. A practical algorithm for l triangulation with outliers. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[59] X Rong Li, Zhanlue Zhao, and Vesselin P Jilkov. Estimators credibility and its measures. In *Proc. IFAC 15th World Congress*, 2002.

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

[61] Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. Simultaneous localization and odometry self calibration for mobile robot. *Autonomous Robots*, 22(1):75–85, 2007.

[62] Martino Migliavacca, Andrea Bonarini, and Matteo Matteucci. RTCAN: a real-time CAN-Bus protocol for robotic applications. In *Informatics in Control, Automation and Robotics (ICINCO), 2013 International Conference on*, 2013.

[63] Martino Migliavacca, Andrea Zoppi, Andrea Bonarini, and Matteo Matteucci. μROSnode: a lightweight ansi c ros client for microcontrollers. In *ROS Developer Conference (ROSCON)*, 2013.

[64] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65, 2009.

[65] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9:1, 2006.

[66] John B Moore. Discrete-time fixed-lag smoothing algorithms. *Automatica*, 9(2):163–173, 1973.

[67] Yoichi Morales, Eijiro Takeuchi, and Takashi Tsubouchi. Vehicle localization in outdoor woodland environments with sensor fault detection. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 449–454. IEEE, 2008.

[68] José Neira and Juan D Tardós. Data association in stochastic mapping using the joint compatibility test. *Robotics and Automation, IEEE Transactions on*, 17(6):890–897, 2001.

[69] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[70] Sanni Siltanen. *Theory and applications of marker-based augmented reality.* 2012.

[71] Jean-Jacques E Slotine and Weiping Li. Applied nonlinear control. *NJ: Prantice-Hall, Englewood Cliffs*, 1991.

[72] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[73] Michael Spivak. A comprehensive introduction to differential geometry, vol. 5. *I (Boston, Mass., 1970)*, 1979.

[74] Simone Stefanini. Estensione del filtro di kalman a gruppi di lie con applicazioni a navigazione e tracking. Master's thesis, Politecnico di Milano, 2013.

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

[76] Niko Sunderhauf and Peter Protzel. Switchable constraints for robust pose graph slam. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1879–1884. IEEE, 2012.

[77] David Tedaldi, Alberto Pretto, and Emanuele Menegatti. A robust and easy to implement method for imu calibration without external equipments. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, page to appear. IEEE, 2014.

[78] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[79] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[80] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment – a modern synthesis. In *Vision algorithms: theory and practice*, pages 298–372. Springer, 2000.

[81] Ken Turkowski. Filters for common resampling tasks. In *Graphics gems*, pages 147–165. Academic Press Professional, Inc., 1990.

[82] Rudolph Van Der Merwe, Eric A Wan, Simon Julier, et al. Sigma-point kalman filters for nonlinear estimation and sensor-fusionapplications to integrated navigation. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pages 1735–1764, 2004.

[83] JF. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira. Geometric approach to strapdown magnetometer calibration in sensor frame. *Aerospace and Electronic Systems, IEEE Transactions on*, 47(2):1293–1306, 2011.

[84] S. Weiss, M.W. Achtelik, M. Chli, and R. Siegwart. Versatile distributed pose estimation and sensor self-calibration for an autonomous mav. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 31–38. IEEE, 2012.

[85] Georgios Zioutas and Antonios Avramidis. Deleting outliers in robust regression with mixed integer programming. *Acta Mathematicae Applicatae Sinica*, 21(2):323–334, 2005.