On the Development of a Generic Multi-Sensor Fusion Framework for Robust Odometry Estimation

Davide A. Cucci[1] Matteo Matteucci[1]
[1] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milano, Italy

Abstract—In this work we review the design choices, the mathematical and software engineering techniques employed in the development of the ROAMFREE sensor fusion library, a general, open-source framework for pose tracking and sensor parameter self-calibration in mobile robotics. In ROAMFREE, a comprehensive logical sensor library allows to abstract from the actual sensor hardware and processing while preserving model accuracy thanks to a rich set of calibration parameters, such as biases, gains, distortion matrices and geometric placement dimensions. The modular formulation of the sensor fusion problem, which is based on state-of-the-art factor graph inference techniques, allows to handle arbitrary number of multi-rate sensors and to adapt to virtually any kind of mobile robot platform, such as Ackerman steering vehicles, quadrotor unmanned aerial vehicles, omni-directional mobile robots. Different solvers are available to target high-rate online pose tracking tasks and offline accurate trajectory smoothing and parameter calibration. The modularity, versatility and out-of-the-box functioning of the resulting framework came at the cost of an increased complexity of the software architecture, with respect to an ad-hoc implementation of a platform dependent sensor fusion algorithm, and required careful design of abstraction layers and decoupling interfaces between solvers, state variables representations and sensor error models. However, we review how a high level, clean, C++/Python API, as long as ROS interface nodes, hide the complexity of sensor fusion tasks to the end user, making ROAMFREE an ideal choice for new, and existing, mobile robot projects.

Index Terms—Sensor Fusion, Localization, Wheeled Robots, Service Robots, Calibration and Identification

1 Introduction

ODOMETRY, i.e., the estimate of a mobile robot position and orientation from proprioceptive measurements, is the first step in the development of autonomous mobile robots and unmanned vehicles. The importance of such activity was clear from early works in mobile robotics [1] and several contributions have been published about improved models for odometry [2], or odometry calibration [3]. When multiple sources of information are available, position and orientation estimation in mobile robots has often been addressed as a problem of multi-sensor data fusion and solved by means of Bayesian filters such as extended Kalman filters and particle filters [4]. The Bayesian filtering approach has also been used for online estimation of calibration parameters, such as, among the others, the systematic and non-systematic components of the odometry error [5], or the GPS latency [6].

Although the effectiveness of these approaches has been proven many times and they are now well established in the literature, people dealing with robot development are still required to write their own ad-hoc implementations to adapt such techniques to the particular application or platform they are developing. But the general case of a modular, kinematic independent, sensor agnostic, self-calibrating, sensor fusion framework is still an open issue, and this is indeed the scenario we targeted in the ROAMFREE (Robust Odometry Applying Multi-sensor Fusion to Reduce Estimation Errors) project. The scientific relevance of this problem is confirmed by the contemporary appearance in the literature of other frameworks for multi-sensor fusion and, in some cases, sensor calibration in mobile robotics: see for instance [7], in which an EKF is employed for pose tracking and sensor self-calibration, and [8] where the sensor fusion problem is modeled by means of a factor graph, in a similar way with respect to the present work.

Modularity enables the re-use of components in different products and prototypes, thus enlarging the share set, reducing costs, in terms of both time and money, and improving overall reliability too. Standardized components have been widely recognized as fundamental in cost effective prototyping, design, and mass production. For instance, in the automotive field, the car platform is often designed to share mechanical and electronic parts among different models so that car manufac-

Regular paper – *Manuscript received November 13, 2013; revised April 30, 2014.*

turer can reduce costs and leverage on platform sharing [9]. In software engineering, software components, generally organized in libraries or frameworks, are re-used among different projects and by several software producers [10].

For the vast majority of robotic applications, it is possible to identify a reasonably small set of common functionalities, which can be implemented in a standard way by modular components. Being odometry one of those, in ROAMFREE we designed a toolkit to be easily used in the development of the odometry system of mobile applications. This goal is significantly more difficult that developing one system that performs odometry by sensor fusion, and should be regarded as a framework to design and program *systems* that perform robust odometry by sensor fusion. An example taken from software engineering could be that of *generic programming* in which "algorithms are written in terms of *to-be-specified-later* types that are then instantiated when needed for specific types provided as parameters" [11].

ROAMFREE sensor fusion library, described in this paper and first introduced in [12][13], aims at the development of a generic framework to provide robust odometry by sensor fusion in mobile robots. Hand coding of a sensor fusion algorithm, is often time consuming and the performance of higher level control and navigation modules are tightly dependent on its localization accuracy. Being able to obtain a result which is comparable to hand coding from a generic algorithm has required some critical design choices which description and discussion is the main contribution of this paper.

The first design choice in ROAMFREE is the definition of *logical sensors* in the form of the type of information they provide about the robot displacement, e.g., absolute pose, linear velocity, acceleration, etc. A second design choice has been the definition of *abstract sensors* in terms of baclk box information sources with a possible displacement with respect to the odometric reference frame of the robot. Finally, tracking and sensor fusion are performed by ROAMFREE through a Maximum-A-Posteriori estimate over the joint probability of robot poses, given the sensor readings, which is represented as a factor graph. While this approach is often employed for the solution of the Simultaneous Localization and Mapping problem, in which it is referred as *bundle adjustment*, its application to inertial navigation has appeared only recently in the literature [14][15]. As we will discuss in details, this approach forms the basis for the modularity of the framework and ultimately enables its sensor-calibration capabilities.

Being able to estimate the robot position and attitude with respect to a world fixed reference frame through multiple sensors requires the calibration of their intrinsic (e.g., biases and distortions) and extrinsic (e.g., relative displacements with respect to the robot frame) parameters. Direct measurement of these quantities is often impractical, or even impossible, and the calibration task is usually performed by means of ad-hoc, hand-tuned procedures especially designed to target the platform in use. The abstract sensor idea, together with the factor graphs non-linear minimization provides also a generic way of performing sensor calibration, both offline and online.

A high level description of the framework is given in Section 2; in Sections 3 and 4 we go more in depth in the techniques employed for the sensors and state variables modeling respectively. According to the factor-graph formalism, each measurement is treated as and error function to be minimized: Section 5 introduces the factor-graph model and the solver used to compute the MAP estimate. One case study about the use of the framework, as well as an experimental evaluation, is reported in Section 6 while Section 7 concludes the paper.

The source code of the ROAMFREE sensor fusion library is released under the *GNU Lesser General Public License (LGPL)* and it is available at[1].

2 Framework Structure Overview

The ROAMFREE sensor fusion library is a flexible and modular framework designed to deliver (i) off-the-shelf position and attitude tracking, (ii) intrinsic, extrinsic, and kinematic parameters self-calibration to mobile robots and unmanned vehicles developers. The framework ships a set of high level sensor models which can be configured in terms of calibration parameters (e.g., distortion and bias coefficients), and geometric displacement on the mobile robot, allowing the end user to precisely describe its robot perceiving architecture instead of coding from scratch the sensor fusion algorithm.

A flexible and modular formulation of the odometry sensor fusion problem allows to deal with an arbitrary number of multi-rate sensors, i.e., various sensors producing readings at different rates, having non-constant frequencies of operation, and possibly producing out of sequence data. The implemented core fusion engine is based on 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; long lags are suited for offline calibration tasks in which the goal is to refine the available estimate of sensor calibration parameters.

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 and characterize them in terms of measurement domain and geometric displacement with respect to the mobile robot kinematic center. We choose not to describe physical models, nor attempt to provide software interfaces for widespread commercial sensors, on the contrary, we follow a *black box* approach focusing on the nature of the information sources.

As a possible example, consider a gyro sensor being part of an inertial measurement unit and a visual odometry algorithm processing images acquired by a calibrated camera: both

1. `http://roamfree.dei.polimi.it`

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, at the level of logical sensor, is a parametrized 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 case of a monocular visual odometry algorithm). We will discuss the *logical sensor* paradigm in details in Section 3.

Another key feature of the ROAMFREE sensor fusion library 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 one exhibiting its own advantages and disadvantages (e.g., Euler angles, unit quaternions, SO(3) manifolds, etc.). When formulating a magnetometer error model, i.e., an equation which relates the predicted sensor measurement given its orientation and the actual Earth magnetic field reading, we do not need to consider 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 an work with arbitrary state variable representation, as we will discuss in details in Section 4.

The core of the ROAMFREE sensor fusion library lies in a software module which 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 logical sensor error models connecting them. Other modules, such as the actual solver algorithms and the outlier rejection module, operate upon this representation, as it will become clear in the next section.

2.1 Functional Description

What follows is a high level description of the functional blocks that implement the sensor fusion tasks, from the point of view of information flow through the framework. While following the description, please refer to Figure 1; details for each of the blocks will be given in later sections.

Data is introduced in the system in the form of timestamped sensor readings; no component is included in the ROAMFREE library to actually *read* data from physical sensors. On the contrary, the user is supposed to implement adapters between hardware or software information sources and the framework. In Section 6 we present very simple and clean approaches to perform this based on ROS [16], which is becoming a *de-facto* standard in robotics research software development.

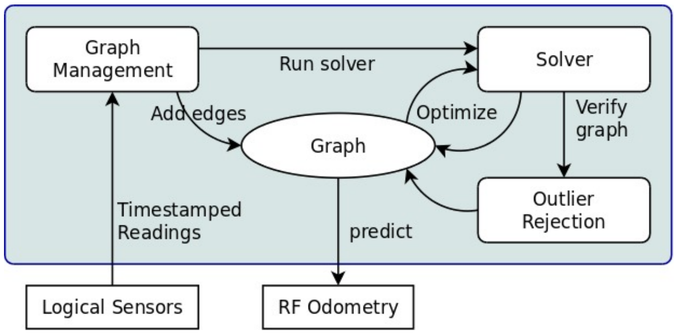


Fig. 1. A simple schema of the ROAMFREE processing triggered by the arrival of new measurements from logical sensors.

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 a 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, i.e., error models (see Section 5).

As new sensor readings are available, the Graph Management component selects the appropriate sensor model and uses it to build a constraint edge in the hypergraph. If needed, e.g., a measurement is newer than the most recent pose in the graph, a new pose node is instantiated and an initial guess is obtained by means of a forward kinematic logical sensor, if available (see Section 3.3). Poses and constraints that are old with respect to the considered time window are discarded and are substituted by a prior constraint, equivalent in a local neighborhood of the current nodes estimate.

Again by means of the Graph Management API, the user invokes a *Solver* to be run on the hypergraph, which now contains the full description of the sensor fusion problem. The Graph Management thus freezes the graph representation, deferring the handling of further sensor readings till the estimation process 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 applies torque to its wheels trying to achieve substantial 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. Indeed,

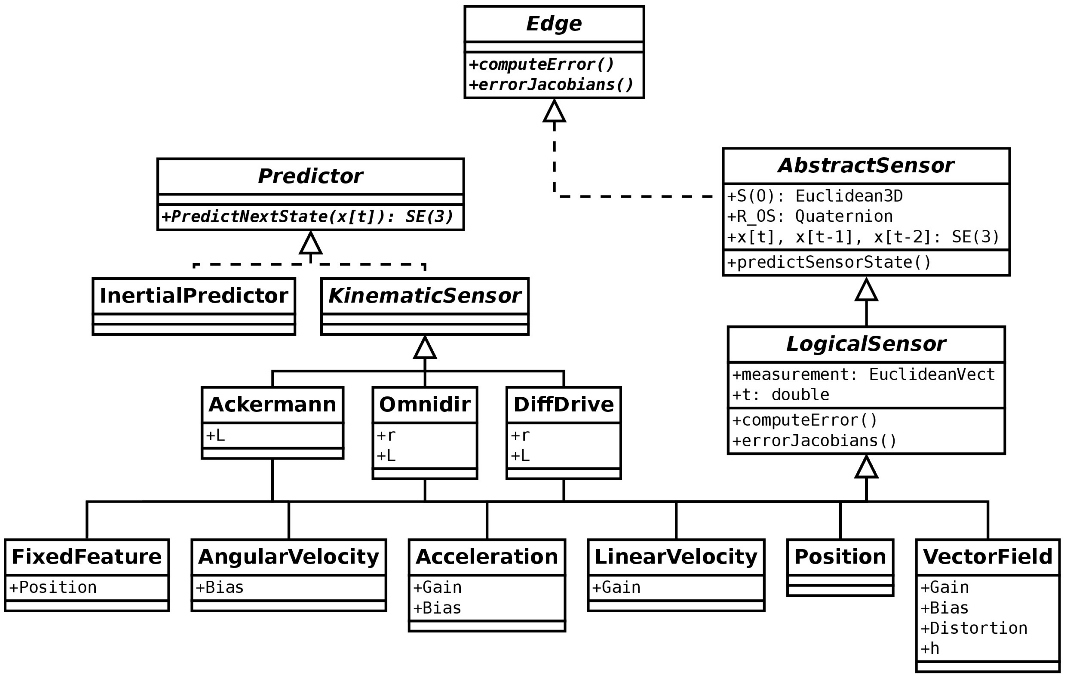


Fig. 2. Outline of the sensor model class hierarchy.

the simplest form of forward kinematic equations assume no slippage, which thus consists in an unmodeled error source.

In the previous scenario, 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. A solution to this problem is to implement consensus heuristics able to detect these situations and selectively disable the outlier sensors edges in the problem graph. This is a very important issue and it is currently subject of active research.

Once the solver has completed its tasks on the factor graph, the 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.

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 a generic model for odometry sensors and odometry sensor fusion that abstracts as much as possible from the nature of the information sources, i.e., the odometry sensors, 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 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 software architecture which implements sensor models (see Figure 2 for reference).

As we have briefly mentioned in Section 2, and as we will see in detail later on, sensor readings compose measurement constraints as edges in the factor graph representing the probabilistic formulation of the information fusion problem. Thus every class in this module implements the generic edge interface. In particular, the key methods here are `computeError()` and `errorJacobians()`, which are the only methods required by solver algorithms.

The next level in the hierarchy consists of *abstract sensors*, which describe the geometric properties of the real sensor placement on the mobile robot, and provide a predictor for the kinematic quantities such as velocities at the sensor reference frame, given the current robot state estimate. Note that in general these are different from the ones in the robot reference frame. Building upon the predictor above, *logical* *sensors* extend this classes providing error models for the actual sensor measurements. More precisely, a measurement predictor, which is function of the kinematic quantities at 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-stage procedure achieves the decoupling between the equations required to handle misaligned and misplaced hardware sensor 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.

3.1 Abstract Sensors

The top level of sensor hierarchy consists in abstract sensors, which give a geometric characterization of information sources. Indeed, the ideal sensor placement in which all sensors position and orientation match the ones of the odometric center of the mobile robot is usually unachievable, or impractical. 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 parameter yield a possibly time varying 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, for which we track the position and orientation with respect to the world fixed frame W. See Figure 3.

Note that at least a rough estimate of these parameters is required to perform the fusion of multiple sensor information properly. To see why, consider for instance a laser rangefinder 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 forward kinematic ones, e.g., when the robot is rotating in-place along its z axis: in this case the rangefinder would report a non-zero linear velocity estimate, since, due to its displacement with respect to the odometric center, the rangefinder moves with respect to the fixed world it perceives while the robot rotates in-place.

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 parameters in the abstract sensors, we can predict the kinematic quantities at the S_i reference frame. More precisely, given the available estimates of the odometric reference frame state, $x^{(O)}(t)$, $x^{(O)}(t-1)$ and $x^{(O)}(t-2)$, and the sensor placement parameters, we derive $\hat{x}^{(S)}(t)$, which is composed by $[\hat{S}_i^{(W)}(t), \hat{R}_{S_i}^{W}(t)]$, the position and orientation of the sensor with respect to the world frame, $[\hat{v}^{(S)}(t), \hat{\omega}^{(S)}(t)]$, its linear and angular velocities and $[\hat{a}^{(S)}(t), \hat{\alpha}^{(S)}(t)]$, the linear and angular accelerations. These quantities characterize the motion of the sensor reference frame with respect to

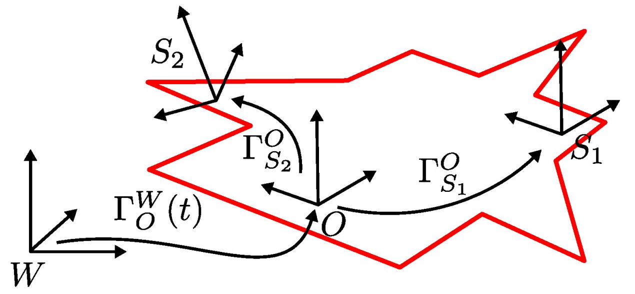


Fig. 3. Reference frames employed in the ROAMFREE sensor fusion library. Abstract sensors S_1 and S_2 are misplaced and misaligned with respect to the Odometric reference frame O.

the world. To derive $\hat{x}^{(S)}(t)$, 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.

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, so to allow the definition of an error function of the form

$$e(t) = \hat{z}\big(t; \hat{x}^{(S)}(t)\big) - z(t) + \eta \tag{1}$$

where η is a zero-mean, Gaussian, noise vector and $e(t) \in \mathbb{R}^n$. The goal of the `computeError()` method is to evaluate this function given the related state variable values. It is possible to see that zero-mean error $e(t)$ is obtained when the prediction matches the actual sensor reading. In this formula we have remarked the dependency of $\hat{z}(t)$ from the sensor state predictor defined by the corresponding abstract sensor in Section 3.1. Equation 1 indirectly relates the robot states $x^{(O)}(t)$, $x^{(O)}(t-1)$ and $x^{(O)}(t-2)$ with the sensor reading at time t. Note that $z(t)$ seldom gives full information on the robot state and, even if this was the case, it is often difficult to invert the sensor model and give closed form expressions for $x^{(O)}(t)$ as a function of $z(t)$. Yet, an estimate of the robot state can be obtained implicitly minimizing $e(t)$ as a function of $x^{(O)}(t)$. In most cases, $e(t)$ is a non-linear function and iterative minimization algorithms are employed. Moreover, it might have multiple local-minima, thus at least a rough initial guess for the state variables and the calibration parameters must be available (see Section 3.3).

The specific form of the error function in Equation 1 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) pose of a world fixed feature (e.g., a landmark) with respect to sensor frame. 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 handle all the hardware sensors and software algorithms commonly employed in mobile robotics pose tracking, e.g. GPS, SLAM, Visual Odometry, gyroscopes, accelerometers, magnetometers, and so on.

Each logical sensor implementation introduces a set of predefined parameters to model sources of distortion, bias or other quantities which have to be known to evaluate the predictor in Equation 1. 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 vector field error model:

$$e(t) = \overbrace{\mathbf{A}\Big(\hat{R}_S^W(t;\mathbf{R}_S^O)\Big)^{-1}\vec{\mathbf{h}}^{(W)} + \mathbf{b}}^{\hat{z}(t)} - z(t) + \eta, \tag{2}$$

where a bold font highlights sensor parameters.

In Equation 2 we made explicit the sensor orientation predictor $\hat{R}_S^W(t)$ dependency on the sensor misalignment parameter $\mathbf{R}_S^O$ (see Section 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 [17], while $\vec{\mathbf{h}}^{(W)}$ have to be set according to the local value of the Earth magnetic field. The ROAMFREE sensor library includes full parametrized implementations of a wide variety of error models for all the physical sensors and processing algorithms commonly employed in mobile robot pose tracking. Moreover, the hierarchical structure of the sensor models allow the end user to easily extend or refine the suite provided.

As a final remark, we present how the evaluation of the `errorJacobian()` method benefits of the hierarchical structure discussed above. This method is required to provide the solver algorithms with a notion of the the error function 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 the robot poses at time t, $t-1$ and $t-2$, the sensor displacement and misalignment parameters, $\mathbf{S}_i$ and $\mathbf{R}_{S_i}^O$, and any other parameter introduced by the current logical sensor. Here we split this evaluation into two steps: first the logical sensors compute the Jacobian of $e(t)$ with respect to $\hat{x}^{(S)}(t)$. Next, the abstract sensor evaluates the Jacobian of $\hat{x}^{(S)}(t)$ with respect to the actual state variables $[x^{(O)}(t), x^{(O)}(t-1), x^{(O)}(t-2), \mathbf{S}_i, \mathbf{R}_{S_i}^O]$. 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, 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 error function is required at the abstract sensor level. Furthermore, the error function can be formulated, and its Jacobian matrix computed, without having to deal with the internal form of the predictors in section 3.1.

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 both the linear and angular velocity of the odometric reference frame can be computed as a function of the sensor readings. Thus, a predictor of the next state $x^{(O)}(t+1)$ can be constructed, given the current state $x^{(O)}(t)$ and the logical sensor reading $z(t)$. These kind of 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 both wheels speed, ω_l and ω_r. The well known forward kinematics equations for such a robot read as:

$$\begin{array}{rcl} \hat{v}_x^{(O)}(t) & = & \frac{\mathbf{r}}{2}\big(\omega_r(t) + \omega_l(t)\big) \\ \hat{\omega}_z^{(O)}(t) & = & \frac{\mathbf{r}}{\mathbf{L}}\big(\omega_r(t) - \omega_l(t)\big) \end{array} \tag{3}$$

where the $\mathbf{r}$ parameter is the wheel radius and $\mathbf{L}$ is the wheel baseline. 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. A simple Euler, or more complex Runge-Kutta, integration scheme yields the required predictor $\hat{x}^{(O)}(t+1)$.

These kind of logical sensors allow to compute reasonable initial guesses for the robot next state before the sensor fusion algorithm is started. Moreover, these sensor 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 refers 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 $x^{(O)}$ 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 $t+1$. It can thus be employed in place of $z(t+1)$ to evaluate the forward kinematic predictor.

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* [18] 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 have they to access variables internal representation, nor have they 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 the sensor fusion algorithms and sensor models formulation from the actual state variable representation. We will discuss an example based on well known unit quaternions in Section 4.1.

State variables can be *fixed* or not[2]. A fixed variable is treated like a constant, i.e., its value is considered to be known 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, **r** and **L**; it is known that their observability depends on the robot trajectory [19]: a 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 the 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 rate 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 [20] (see Section 4.2 for details).

4.1 Variable Domains

As previously introduced, each state variable has its own domain. Well known examples of variable domains which

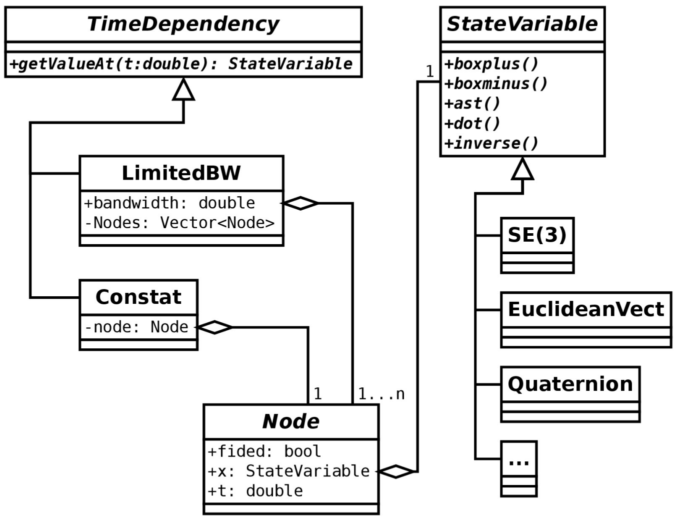


Fig. 4. The state variables class hierarchy.

are not Euclidean are unit quaternions, often employed to represent 3-DOF rotation, and elements belonging to the space of 6-DOF rigid transformations, $SE(3)$. 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 quaternion in EKFs: during updates, unless special care is taken by means of *ad-hoc* methods, e.g., Lagrange multipliers [21], the norm of the quaternions eventually diverges from 1 and normalization has to be performed. Furthermore, due to overparametrization, the full state covariance matrix is always ill-conditioned.

In our work we follow the idea in [18] and define a state variable interface which requires a set of operators to be implemented. These allow both the error function to be evaluated and the solver algorithm to perform state estimation without the need to know or to handle the internals of state variable representations. These operators are:

1) $\boxplus : \mathcal{M} \times \mathbb{R}^n \rightarrow \mathcal{M}$
2) $\boxminus : \mathcal{M} \times \mathcal{M} \rightarrow \mathbb{R}^n$
3) $* : \mathcal{M} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, default action on $\mathbb{R}^n$
4) $^{-1} : \mathcal{M} \rightarrow \mathcal{M}$, inverse
5) $(\cdot) : \mathcal{M} \times \mathcal{M} \rightarrow \mathcal{M}$, composition

The $\boxplus$ operator applies a local, Euclidean, increment to the non-Euclidean variable belonging to the manifold $\mathcal{M}$ and it is employed by the sensor fusion algorithm to modify state variables ensuring the consistency of their internal representation. The $\boxminus$ operator can be seen as an inverse of the previous operator, and $x \boxminus y$ gives the element $\delta \in \mathbb{R}^n$ such that $x \boxplus \delta = y$. The $*$ operator performs the manifold default action on a real, Euclidean, vector, e.g., the application of

2. 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.

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. For instance, note that the expression $x \boxminus (x \boxplus \delta) = \delta$ cannot hold for every δ, since x *resembles* $\mathbb{R}^n$ only locally. Refer to [18] for mathematical details.

To clarify how these operators can be implemented in a practical case, we develop here an example based on unit quaternions, which are the default choice to represent 3-DOF orientations in the ROAMFREE sensor fusion library. Consider a quaternion $q = [q_w, q_x, q_y, q_z]$ 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

$$\dot{q} = \frac{1}{2} Q [0, \omega_x, \omega_y, \omega_z]^T + O(|\omega|^2) \tag{4}$$

where Q is the matrix representation of the quaternion product operator and it is given by

$$Q = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix}. \tag{5}$$

To build the $\boxminus$ operator, we observe that, once we have truncated Equation 4 to the first order and we have applied an Euler integration scheme, 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. Note that this holds 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.

Next, we derive the $q * x$, $x \in \mathbb{R}^3$ expressions, in which the x vector is rotated by q. The well known formulas to construct a rotation matrix from a quaternion q are:

$$R(q) = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_w q_y + q_x q_z) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(-q_w q_x + q_y q_z) \\ 2(-q_w q_y + q_x q_z) & 2(q_w q_x + q_y q_z) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{6}$$

and thus we can define $q * x$ as $\tilde{x} = R(q)x$, where the usual matrix product takes place.

The remaining two operators comes from the unit quaternion algebra and it holds that $q^{-1} = [q_w, -q_x, -q_y, -q_z]$ and that $q_1 \cdot q_2 = Q_1 q_2$, see Equation 5.

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, we decouple the internal representation of meaningful quantities from their manipulation.

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.

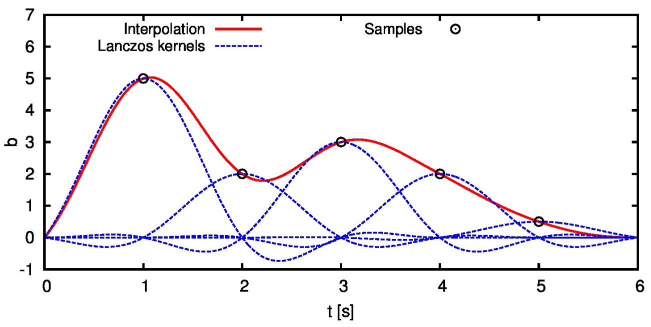


Fig. 5. Example of Lanczos resampling. Five samples are interpolated to produce a parameter signal whose maximum bandwidth is 0.5 Hz.

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. A basic error model in this case would be:

$$e(t) = \overbrace{\omega(t) + \mathbf{b}(t)}^{\hat{z}(t)} - z(t) + \eta, \tag{7}$$

in which we have assumed that the gyroscope observes the true angular speed, up to a bias, ignoring any other error sources such as axes non-orthogonality. Since the bias $\mathbf{b}$ is not directly observable, if we lack for other observations regarding $\omega(t)$, coming from other sensors, it is easy to see that $||e(t)||$ can be arbitrarily reduced by selecting proper values of $\mathbf{b}(t)$, thus compromising the information carried by $z(t)$. The simplest solution here 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. The fusion engine thus would try to estimate the unknown value of $\mathbf{b}(t)$ assuming that $\mathbf{b}(t_1) = \mathbf{b}(t_2)$, $\forall t_1, t_2$.

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. Another possible choice 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 a which of the 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 5):

$$L(t) = \begin{cases} a\frac{sin(2\pi ft)sin\frac{2\pi f}{a})}{(2\pi ft)^2} & if - a < 2\pi ft < a \\ 0 & otherwise \end{cases} \quad (8)$$

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:

$$\mathbf{b}(t) = \sum_{k=\lfloor 2ft \rfloor - a+1}^{\lfloor 2ft \lfloor +a} \mathbf{b}_k L(2\pi ft - k), \quad (9)$$

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 7 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, a general interpolation scheme for variables belonging to an arbitrary manifold has not been implemented yet and 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 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.

5 The Core Fusion Engine

The ROAMFREE sensor fusion library is based on state of the art, non-linear, Bayesian inference techniques on a factor graph representing the joint probability distribution of the state variables, given the sensor measurements [22]. Here Gauss-Newton/Levenberg-Marquardt [23] optimization algorithms are employed whose goal is to find the Maximum A Posteriori (MAP) of this probability distribution, i.e., the configuration of state variables which maximize their joint probability, given all the sensor measurements. These formulation has become popular in the Simultaneous Localization and Mapping (SLAM) and photogrammetric communities under the name of *Bundle Adjustment* [24] and it has been effectively applied to large scale and/or online, 3D reconstruction, problems in which both the camera egomotion and the world-attached features have to be tracked. More recently, factor graph approaches to the odometry and parameter self-calibration problems appeared in the literature [14][15]. Despite the excellent performances reported for state-of-the-art methodologies, few solutions exist that deliver multi-sensor

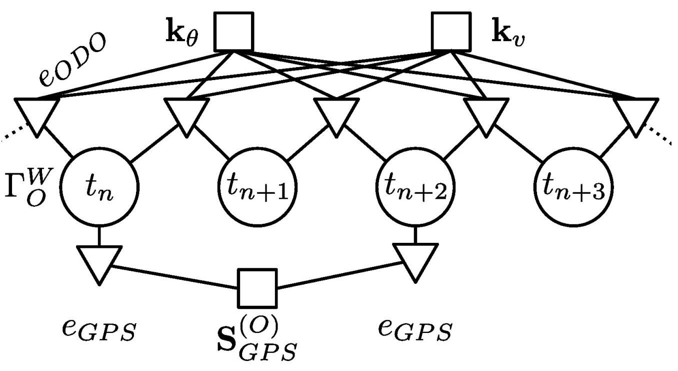


Fig. 6. An instance of the pose tracking factor graph with four pose vertices $\Gamma_O^W(t)$ (circles), odometry edges e_{ODO} (triangles), two shared calibration parameters vertices $\mathbf{k}_v$ and $\mathbf{k}_\theta$ (squares), two GPS edges e_{GPS} and the GPS displacement parameter $\mathbf{S}_{GPS}^{(O)}$.

pose tracking and calibration as an *off-the-shelf*, flexible and modular component. In this section we discuss the key features of the ROAMFREE Graph Management and Solver components, which form the basis for the desirable properties above.

The Graph Management component maintains a hypergraph in which 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 the relevant sensor calibration parameters. For instance, consider an odometry measurement obtained from the wheel speed readings in a differential drive robot at time t. This information defines a hyperedge involving the robot poses at time $t-1$ and t, the wheel radius and baseline distance parameters. This edge constraints the difference of the two poses according to the forward kinematics of the robot. In Figure 6 it is possible to see a more complex example involving odometry constraint 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, $\mathbf{S}_{GPS}^{(O)}$.

When a new sensor reading is available, the Graph Management component is responsible of instantiating a proper logical sensor edge and inserting it into the factor graph as it is incident to the pose and sensor parameter nodes required to evaluate the state likelihood (i.e., the logical sensor error function). In particular, the logical sensor traits specify the *order* of the associated error function, i.e., if it constraints only robot pose and/or orientation (order 0), or if it refers to velocities and accelerations (order 1 and 2), and the calibration parameters, as long as the dimensions of the error and noise vectors. The proper pose nodes are then chosen according to the logical sensor order and measurement timestamp. In case the sensor reading is newer with respect to the latest pose available in the graph, a new node has to be instantiated. The Graph Management component thus checks if the current logical sensor implements the `Predictor` interface. If so, the `predictNextState()` method is called to compute an initial guess for the new pose node. Otherwise, the measurement handling is deferred. 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 constraints to be removed as well, implying information loss. To avoid this, old nodes are marginalized and an linear constraint which is locally equivalent with respect to the removed nodes and constraints it is inserted over their Markov blanket, as described in [25].

The advantages of the factor graph formulation for the pose and parameter tracking problem are many; first of all, it allows for an arbitrary number of sensors to be handled in a modular and independent way: different sensor models implement the abstract hyperedge interface and they are handled uniformly as they are inserted into the graph. This also implies that sensors can be dynamically turned on and off online. Moreover, out-of-order measurements, i.e., sensor readings which are reported to the Graph Management module with 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 edge. Note that, even if sensor reading edges constraint past nodes, they still contribute to the refinement of the most recent pose estimate through the other constraints already present in the graph. In a similar way, it is possible to deal with arbitrary and non-constant reading rates. Moreover, from the estimation quality point of view, it has been argued that the factor graph formulation is more accurate, and, in certain circumstances, even less computationally expensive than traditional EKFs [26].

As it has been presented in Section 3 and 4, sensor models and state variables conceal their internals under nodes and hyperedges interfaces. Thus, the specific algorithm which solves the MAP problem is independent from their specific formulation. The guideline adopted here is to decouple variable and sensor modeling from the problem solution tasks. Abstract interfaces for nodes and edges are defined such that a generic, non-linear, optimization algorithm can deal with arbitrary nodes and edges without knowing their internal structure and functioning. At the highest level, edges, and thus logical sensors, implement the `computeError()` method, which associates a likelihood to the current configuration of involved nodes, given the sensor reading and error model. An additional `errorJacobians()` method may be added to provide the solver with a notion of the error function direction of steepest descent, avoiding its computation with numerical schemes such as finite differences. Finally, nodes hide their internal representation to the solver algorithms by means of the $\boxplus$ and $\boxminus$ operators introduced in Section 4.1. Consider for example a Gauss-Newton optimization algorithm, which computes an approximation of the MAP attempting to minimize the sum of the squared error vectors, weighted by sensor information matrices. To do so, it iterates through all the edges in the hypergraph asking error vectors and Jacobian matrices evaluation. It then construct the Hessian matrix of the linearized system and solves the resulting quadratic form yielding values for the Euclidean increments, which are then applied to the variables by means of the $\boxplus$ operator and the process is repeated till termination criteria occur.

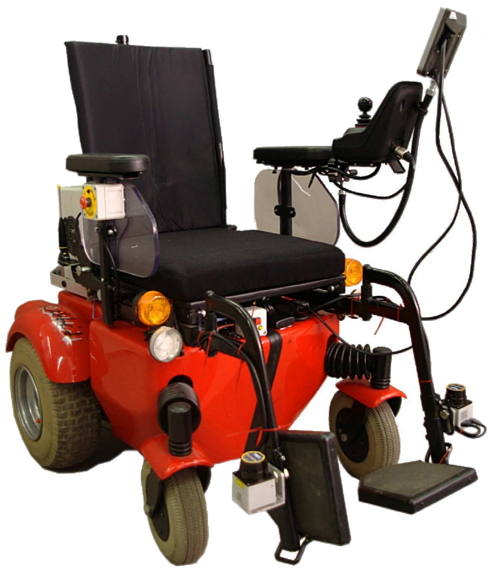


Fig. 7. The LURCH autonomous wheelchair. The two Hokuyo URG-04LX are visible near the footrests, while the Prosilica is mounted behind the seatback.

As for the implementations of the solver algorithms we rely on the g^{2}o [27] software library, a general framework for least squares optimization, fine tuned to exploit the sparsity of factor graph optimization problems such as the ones considered in this work. At the present stage of development, three solver algorithms are available: Gauss-Newton, LevenbergMarquardt and Peconditioned Conjugate Gradients. Nevertheless, many sensor fusion algorithms, such as an Extended Kalman Filter, or incremental smoothing, such as iSAM2 [28], can be formulated such that they rely only on the available primitives and implemented as new MAP solvers without changes in the rest of the architecture.

6 Experimental Evaluation

In this section we show how the ROAMFREE sensor fusion library can be employed to perform the pose tracking and odometry calibration tasks for the LURCH [29] autonomous wheelchair (see Figure 7). The key aspect of the deployment of the sensor fusion node will be discussed assuming that ROS is employed to distribute and pre-process sensor readings. Along with the description of the software architecture, we will consider an indoor benchmark scenario in which fiducial markers are present in the environment and we will discuss pose tracking and calibration results. We will show how the

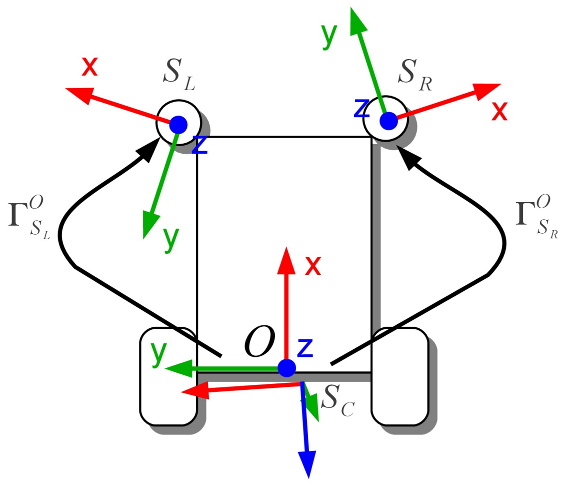


Fig. 8. The LURCH sensor frames: S_L and S_R placed at the two Hokuyo range-fiders, and S_C at the Prosilica.

same software module can be employed to perform offline parameter self-calibration, relying on sensor readings only, and online pose tracking, employing the determined estimates.

The LURCH autonomous wheelchair 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. Moreover, a Prosilica GC1020 is mounted behind the seatback and looking backward, capturing frames at 10 Hz. As fiducial marker become visible, they give a notion of the camera position with respect to the world. Finally, wheel speed is estimated at 50 Hz rate from low resolution encoders. A sketch of the reference frames is depicted in Figure 8.

While wheel encoder sensor readings can directly feed the ROAMFREE sensor fusion library, the laser range-finders point clouds and the camera images are not directly handled by the framework. However, 2D estimates for the linear and angular velocities can be obtained by processing successive laser scans with an Iterative Closest/Corresponding Point (ICP) algorithm [30], which also return a measurement of the estimates uncertainty. In the considered architecture, two identical ROS nodes, one for the left Hokuyo and one for the right one, subscribe to the respective scan topic and publish velocity estimates. Moreover, another node subscribes to the camera image topic and employs the ALVAR library [31] to track fiducial markers visible in the current frame, publishing their relative position with respect to the camera. In this case no uncertainty measure is computed by the tracking library. Here we assume that it increases with the square of the marker distance. Note that the two modules above are not part of the framework and they come as off-the-shelf ROS components.

In the following we will setup a `rospy` node that configures logical sensors, subscribes to measurement topics, delivers them to the Graph Management module (See Section 5) and operates the main estimation loop.

In Listing 1 we begin instantiating a wrapper object including all the components required to solve a pose tracking problem. By means of its constructor arguments we specify key properties such as the length of the fixed-lag window, in seconds, the timestamp of the first pose and its initial value, as long as the solver algorithm for the optimization problem. We employ a Gauss-Newton solver for online tracking, because of its deterministic execution time. Conversely, in offline calibration runs we choose Levenberg-Marquardt, which, adjusting the step size, it is able to prevent divergence and better handles irregular error function landscapes.

```
RF = ROAMFREE(
  fixedLagLenght = L,
  t0 = rospy.get_time(),
  x0 = [0, 0, 0, 1, 0, 0, 0],
  solver = SolverMethod.GaussNewton
)
```

Listing 1. Instantiation of the main ROAMFREE object

```
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', [R_guess], fixed=True) # wheel radius
RF.addParameter(ParameterTypes.Euclidean1D,
   'Odo_L', [L_guess], fixed=True) # ... and baseline
```

Listing 2. Configuring the differential drive kinematics logical sensor for online tracking. It is the master sensor, i.e., the Forward Kinematic one which triggers pose nodes to be instantiated when new reading are available.

Next, we configure a Differential Drive logical sensor to handle wheel encoder readings (see Listing 2). This sensor refers to the odometric center of the robot, it has null displacement and misalignment abstract sensor parameters. Moreover, it sports two constant kinematic parameters, **r** and **L**, i.e., the wheel radius and the robot baseline. During offline calibration we set the `fixed` flag (See Section 4) to `False` and we provide initial guesses obtained by means of direct inspection on the robot; then, we consider long time lags and employ all the collected sensor readings to simultaneously estimate robot poses and parameters without the need of an external ground truth source. Conversely, in the online pose tracking case we fix these parameters and employ their calibrated value. Note that this is the kinematic master sensor, i.e., the one which causes new pose nodes to be inserted into the graph as new sensor readings become available (see Section 5). Other

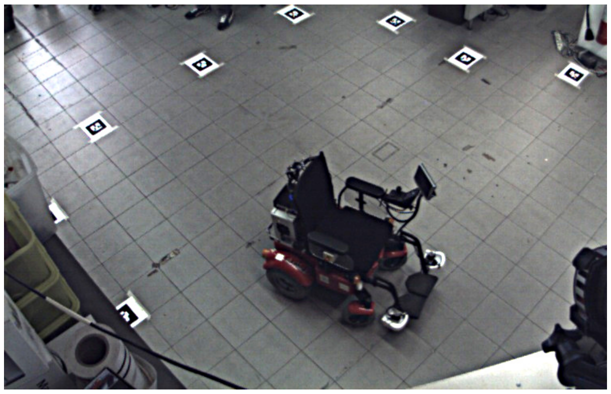


Fig. 9. Experimental setup

sensors instead have to pick the existing pose whose timestamp is nearest to the current reading.

The ICP scan-matching nodes estimate the rigid transformation needed to overlap two successive laser scans, which can be interpreted as a linear and angular velocity measure, given the sensor operating frequency. In Listing 3 we setup a generic odometer logical sensor to handle the ICP scan-matching output. The misplacement and misalignment of this sensor cannot be neglected: while we are quite confident in our knowledge of the laser displacement, we have only a rough guess for the laser yaw. Thus, in offline calibration runs, we configure the misalignment parameter as to be estimated setting to `False` its `fixed` flag.

```
RF.addSensor('Hokuyo_left', master=False,
   type=SensorTypes.GenericOdometer)

RF.setSensorDisplacement('Hokuyo_left',
   [0.75, 0.25, 0.0], fixed=True)
RF.setSensorMisalignment('Hokuyo_left',
   [0.0, 0.0, 90.0], fixed=False) #looks to the left
```

Listing 3. Configuring the logical sensor to handle the left Hokuyo scan-matching readings. The right ones are handled in a similar way.

Next, we have to configure the logical sensors that allows to handle fiducial markers position readings. As it happens with image features in visual SLAM algorithms, the position of the markers with respect to the world, encoded in the calibration parameter $\mathbf{F}^{(W)}$, it is not known and it has to be determined online. However, the considered case is simpler with respect to the visual SLAM one since each fiducial marker encodes a different identifier which allows to distinguish between them, ultimately eliminating the data association problem. In the considered scenario, each time a new marker is detected by the Alvar tracking library, a new logical sensor is instantiated and its calibration parameter is initialized with the current robot pose (see Listing 4). Note that, while we need multiple logical sensor to handle the fiducial markers, all of the sensor readings come from the same physical sensor, i.e., the Prosilica camera, thus we make the fiducial marker sensors *share* the misplacement and misalignment calibration parameters with the camera. These parameters are instantiated separately, as it has been done in the range-finders case, and the misalignment one is set to be estimated in calibration runs.

```
RF.addSensor('FM_'+str(id), master=False,
   type=SensorTypes.FixedFeaturePosition) #e.g. FM_5

RF.addParameter(ParameterTypes.Euclidean3D,
   'FM_'+str(id)+'Fposition', pose, fixed=False)

RF.shareGeometricParameters('FM_'+str(id), 'Camera')
```

Listing 4. Configuring the logical sensor to handle fiducial marker position readings

Once all the required logical sensors have been configured, we define the callback functions that, according to the publish/subscribe paradigm implemented in ROS, will be invoked as soon as new sensor reading messages are available. In Listing 5 we present an example of a callback to handle the output of the ICP scan-matching nodes, which are supposed to be available as standard ROS `nav_msgs/Odometry` messages. The `addMeasurement` method, which is part of the Graph Management module API, is the only method that has to be called to feed the sensor fusion library.

```
def laser_callback(msg):
  z = [ msg.twist.twist.linear.x,
        msg.twist.twist.linear.y,
        0.0,
        0.0,
        0.0,
        msg.twist.twist.angular.z
        ]
  T = timeFromROSheader(msg)

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

rospy.Subscriber("/laser_odometry_left",
  nav_msgs/Odometry, laser_callback)
```

Listing 5. Defining a callback function to handle laser odometry readings available as standard ROS messages.

```
# try to run at constant frequency, i.e. 10 Hz
r = rospy.Rate(10)

while not rospy.is_shutdown():
  # run 5 solver iterations
  pose = RF.estimate(iterations=5)

  # publish pose transformation
  TF.sendTransform(pose, rospy.Time.now(),
    "base_link", "world")

  r.sleep();
```

Listing 6. Running estimation and publishing results.

Finally, we review the main loop for online pose tracking, which is typically run at constant rate. The pose estimate can be made available to the rest of the architecture as a `tf`, transform as shown in Listing 6. Note that incoming sensor

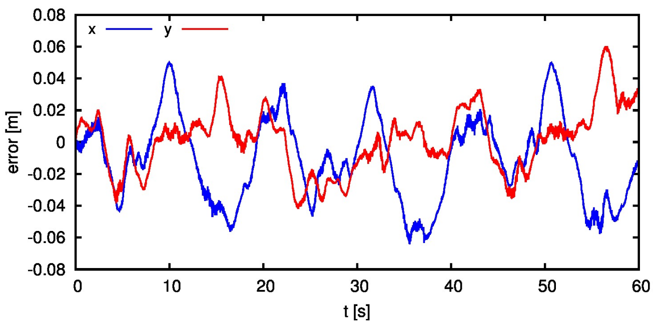


Fig. 10. Final LURCH X-Y position error with respect to ground truth for the calibration run.

readings are handled by callback functions in different threads. Concurrent access to the factor-graph is handled internally by the Graph Management component.

Next we discuss pose tracking and sensor parameter calibration benchmarks for the described software architecture. We consider an area of approximately 25 m^2 and we disseminate on the floor 14 markers of size 18×18 cm (see Figure 9). An Optitrack motion capture system, which reaches millimeter level tracking accuracy, is available as a ground truth source for both the markers and the wheelchair pose.

We first perform the calibration of the unknown sensor parameters, which in the considered case are: the z components of the laser misalignment $\mathbf{q}^O_{S_L}$ and $\mathbf{q}^O_{S_R}$, the 3-DOF camera misalignment $\mathbf{q}^O_{S_C}$, the wheel radius $\mathbf{r}$ and the baseline $\mathbf{L}$, along with the position of the fiducial markers. For these parameters, except for the marker positions, we provide initial guesses obtained by direct inspection on the robot. To collect sensor readings for the offline calibration runs we manually drive the robot along eight-shaped paths inside the marker area. The `rospy` node waits till a 60 s fixed-lag-windows has been filled and then triggers the estimation process. We first determined the camera misalignment, then the wheel radius and baseline, and finally the laser range-fiders yaw. Each calibration problem had approximately 5000 constraint edges and could be solved by menas of levenberg-Marquardt in less than 2 s. In Figure 10 we plot the LURCH position error with respect to the Optitrack ground truth at the end of the calibration process, while in Table 1 we list the estimated marker positions and their true value. The results for the calibration parameters can be seen in Table 2.

Unfortunately, it is very difficult to provide ground truth values for sensor calibration parameters in real world experiments, especially regarding 3-DOF rotations. However, note that the orientation of the camera with respect to the robot has to be precisely determined, especially regarding the pitch, to be able to correctly determine the position of the markers. From pose tracking and marker position estimate results it is possible to see that outstanding, centimeter level, position accuracy has been achieved while simultaneously being able to calibrate unknown sensor parameters.

		estimate [m]	GT [m]	error [cm]	distance [cm]
M1	x	3.444	3.400	4.378	9.709
	y	−0.320	−0.238	−8.299	
	z	0.025	0.000	2.493	
M2	x	2.596	2.553	4.300	10.945
	y	−1.188	−1.089	−9.897	
	z	0.018	0.000	1.833	
M3	x	1.757	1.707	5.033	14.155
	y	−2.062	−1.931	−13.170	
	z	0.013	0.000	1.262	
M4	x	0.700	0.741	−4.113	4.377
	y	−1.863	−1.860	−0.365	
	z	0.014	0.000	1.450	
M5	x	0.107	0.110	−0.262	2.498
	y	−1.203	−1.225	2.148	
	z	0.012	0.000	1.248	
M6	x	−0.947	−0.946	−0.114	2.908
	y	−1.012	−1.012	−0.004	
	z	0.029	0.000	2.906	
M7	x	−1.659	−1.582	−7.715	8.402
	y	−0.400	−0.370	−3.089	
	z	0.012	0.000	1.233	
M8	x	−1.800	−1.728	−7.203	8.364
	y	1.096	1.066	3.004	
	z	0.030	0.000	3.008	
M9	x	−1.272	−1.266	−0.556	8.580
	y	2.097	2.015	8.242	
	z	0.023	0.000	2.316	
M10	x	−0.256	−0.265	0.896	12.925
	y	2.391	2.263	12.852	
	z	0.010	0.000	1.037	
M11	x	0.841	0.801	3.998	15.056
	y	2.673	2.531	14.207	
	z	0.030	0.000	2.978	
M12	x	1.823	1.724	9.849	11.084
	y	2.404	2.365	3.980	
	z	0.032	0.000	3.166	
M13	x	2.714	2.610	10.395	10.721
	y	1.566	1.560	0.627	
	z	0.025	0.000	2.548	
M14	x	3.599	3.473	12.634	13.274
	y	0.747	0.725	2.209	
	z	0.034	0.000	3.422	

TABLE 1
Final marker position estimate and ground truth for the calibration run.

Next we consider the online pose tracking case in which the estimated values for the sensor calibration parameters are employed, with the exception of the fiducial marker positions, which are tracked online, as it happens in SLAM algorithms. Here we consider a 10 s fixed-lag window length, and older nodes are marginalized as described in Section 5, so that in any given time window we do not have observations for every marker. We run the main estimation loop at 10 Hz for 240 s. The resulting position error is depicted in Figure 11. It is possible to see that centimeter level accuracy is obtained and that the overall position error is bounded despite the old constraint marginalization procedure.

In this section have presented an overview of how very complex sensor fusion tasks can be setup and handled in a clean,

		estimate	initial guess
r		0.159	0.150
L		0.545	0.500
$\mathbf{q}^{O}_{S_C}$	w	$+0.3641$	$+0.000$
	x	-0.5837	$+0.000$
	y	-0.5940	$+0.000$
	z	$+0.4108$	$+1.000$
$\mathbf{q}^{O}_{S_L}$	z	$+0.5966$	$+0.7071$
$\mathbf{q}^{O}_{S_R}$	z	-0.6659	-0.7071

TABLE 2
Final parameter estimates and provided initial guesses for the calibration run.

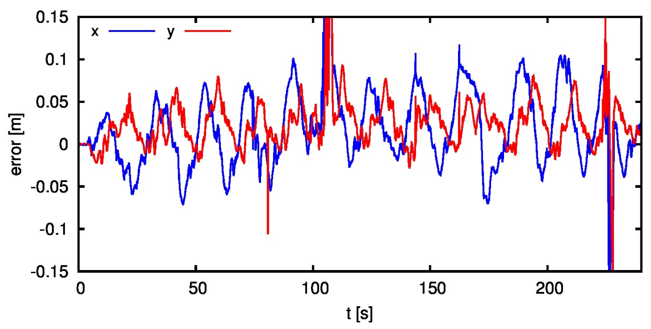


Fig. 11. Final X-Y position error with respect to ground truth for the online run.

simple and abstract way by means of the ROAMFREE sensor fusion library. All the internals of the sensor fusion problem and handling are hidden behind a clean Python API, delivering a complete, off-the-shelf, pose tracking and sensor calibration framework. Moreover, we have discussed benchmarks for the pose tracking and sensor parameter calibration case which showed that remarkable performances are achievable and no application dependent heuristic or tweak is required.

7 Conclusions

In this paper we have presented the design choices we made in ROAMFREE, a generic framework for the development of odometry systems through sensor fusion. Although several methods for odometry and pose tracking have been proposed in the literature, no open-source framework for out-of-the-box odometry and sensor self-calibration was available.

As in any generic framework, ROAMFREE modularity and flexibility come at a cost. The development of such a system required the design of a complex software architecture achieving the decoupling among 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 the 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 sensor provide a fundamental abstraction which makes the system independent with respect to the actual platform and hardware sensor employed. On the other hand, the *black-box* assumption prevents the solver to deal with the internals of the actual sensor or processing algorithm.

Consider for instance a visual odometry system. These algorithms usually track simultaneously the camera egomotion and a set of world-fixed features. Their output could be used to drive a linear and angular velocity logical sensor that, along with the other ones available in the architecture, would contribute to the final pose estimate. However, the pose estimate obtained with the full sensor set will in general be different from the one that is determined internally by the visual odometry algorithm. Because of the *black-box* assumption, no general mechanism is available to overcome this potential source of inconsistency. It is subject of active research to determine how the ROAMFREE pose estimate could be employed to provide general feedback loops to the 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 sensor set. This would further increase he overall complexity of the framework; hiding this complexity to the end user, which could also be the logical sensor developer, will be the next challenge.

As a further extension, we note that the developed sensor fusion library it is already able to solve the *full* and the *online* SLAM problems. A simplified example of such application was presented in Section 6, where exteroceptive logical sensors were employed that measured the relative position of a world attached feature, i.e., the fiducial markers. In the calibration run we were able to precisely determined the map of the environment, in this case consisting in the fiducial markers positions, delimiting the robot operation area. More complex cases could be considered in which the fiducial marker were replaced with scale invariant features detected in camera images, as it happens in visual SLAM algorithms, or with laser scan associated with robot poses.

Acknowledgments

Thi work has been partially supported by the Italian Ministry of University and Research (MIUR) through the PRIN 2009 project “ROAMFREE: Robust Odometry by Applying Multi-sensor Fusion to Reduce Estimation Errors”, and by Regione Lombardia and MIUR through the SINOPIAE project.

References

[1] J. Borenstein, H. Everett, and L. Feng, “Where am i? sensors and methods for mobile robot positioning,” *University of Michigan*, vol. 119, p. 120, 1996. 1

[2] L. Ojeda and J. Borenstein, “Methods for the reduction of odometry errors in over-constrained mobile robots,” *Autonomous Robots*, vol. 16, pp. 273–286, 2004. 1

[3] A. Martinelli, "The accuracy on the parameter estimation of an odometry system of a mobile robot," in *Proceedings of International Conference on Robotics and Automation (ICRA 2002)*, 2002, pp. 1378–1383. 1

[4] Z. Chen, "Bayesian filtering: From kalman filters to particle filters, and beyond," *Statistics*, vol. 182, no. 1, pp. 1–69, 2003. 1

[5] A. Martinelli, N. Tomatis, and R. Siegwart, "Simultaneous localization and odometry self calibration for mobile robot," *Autonomous Robots*, vol. 22, no. 1, pp. 75–85, 2007. 1

[6] D. Bouvet and G. 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*. IEEE, 2000, pp. 2525–2530. 1

[7] S. Weiss, M. 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*. IEEE, 2012, pp. 31–38. 1

[8] H.-P. Chiu, S. Williams, F. Dellaert, S. Samarasekera, and R. Kumar, "Robust vision-aided navigation using sliding-window factor graphs," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 46–53. 1

[9] M. Muffatto, "Introducing a platform strategy in product development," *International Journal of Production Economics*, vol. 6061, pp. 145–153, 1999. 1

[10] G. Heineman and W. Councill, *Component-Based Software Engineering: Putting the Pieces Together*. Addison Wesley Professional, 2001. 1

[11] D. R. Musser and A. A. Stepanov, "Generic programming," in *Symbolic and Algebraic Computation*, ser. Lecture Notes in Computer Science, P. Gianni, Ed. Springer Berlin Heidelberg, 1989, vol. 358, pp. 13–25. 1

[12] D. A. Cucci and M. 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*, 2013, pp. 361–368. 1

[13] ——, "Position tracking and sensors self-calibration in autonomous mobile robots by gauss-newton optimization," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, p. to appear. 1

[14] V. Indelman, S. Williams, M. Kaess, and F. Dellaert, "Information fusion in navigation systems via factor graph based incremental smoothing," *Robotics and Autonomous Systems*, 2013. 1, 5

[15] R. Kümmerle, G. Grisetti, and W. Burgard, "Simultaneous parameter calibration, localization, and mapping," *Advanced Robotics*, vol. 26, no. 17, pp. 2021–2041, 2012. 1, 5

[16] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009. 2.1

[17] J. 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*, vol. 47, no. 2, pp. 1293–1306, 2011. 3.2

[18] C. Hertzberg, R. Wagner, U. Frese, and L. SchröDer, "Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds," *Information Fusion*, vol. 14, no. 1, pp. 57–77, 2013. 4, 4.1, 4.1

[19] A. Censi, L. Marchionni, and G. Oriolo, "Simultaneous maximum-likelihood calibration of odometry and sensor parameters," in *Robotics and Automation (ICRA), 2008 IEEE International Conference on*. IEEE, 2008, pp. 2098–2103. 4

[20] K. Turkowski, "Filters for common resampling tasks," in *Graphics gems*. Academic Press Professional, Inc., 1990, pp. 147–165. 4

[21] R. Van Der Merwe, E. A. Wan, S. 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*, 2004, pp. 1735–1764. 4.1

[22] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, 2001. 5

[23] P. E. Gill and W. Murray, "Algorithms for the solution of the nonlinear least-squares problem," *SIAM Journal on Numerical Analysis*, vol. 15, no. 5, pp. 977–992, 1978. 5

[24] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *Vision algorithms: theory and practice*. Springer, 2000, pp. 298–372. 5

[25] N. Carlevaris-Bianco and R. M. Eustice, "Generic factor-based node marginalization and edge sparsification for pose-graph slam," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5748–5755. 5

[26] H. Strasdat, J. Montiel, and A. J. Davison, "Real-time monocular slam: Why filter?" in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2657–2664. 5

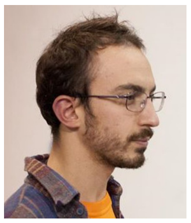
[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g^2o: A general framework for graph optimization," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3607–3613. 5

[28] M. Kaess, S. Williams, V. Indelman, R. Roberts, J. Leonard, and F. Dellaert, "Concurrent filtering and smoothing," in *Information Fusion (FUSION), 2012 15th International Conference on*, 2012, pp. 1300–1307. 5

[29] R. Blatt, S. Ceriani, B. Dal Seno, G. Fontana, M. Matteucci, and D. Migliore, "Brain control of a smart wheelchair," in *10th International Conference on Intelligent Autonomous Systems*, 2008. 6

[30] A. Censi, "An accurate closed-form estimate of icp's covariance," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 3167–3172. 6

[31] S. Siltanen, *Theory and applications of marker-based augmented reality*, 2012. 6



Davide Antonio Cucci Laurea degree 2008, Laurea Specialistica degree 2011 is a Ph.D. Candidate at Politecnico di Milano. He was involved in the PRIN 2009 grant "ROAMFREE: Robust Odometry Applying Multi-sensor Fusion to Reduce Estimation Errors" as main developer of the ROAMFREE sensor fusion library. During the QUADRIVIO project, founded by FILAS S.p.A. and Regione Lazio (Italy), he was responsible for the development of the localization and autonomous navigation modules for an all-terrain mobile robot. His main interests lie in information fusion by means of optimization techniques, with main applications in mobile, autonomous robot tracking and localization.

Matteo Matteucci Laurea degree 1999, M.S. 2002, Ph.D. 2003 is Assistant Professor at Politecnico di Milano. Divided between Robotics and Algorithms for Machine Learning, in 2002 he received a Master of Science in Knowledge Discovery and Data Mining at Carnegie Mellon University (Pittsburgh, PA), and in 2003 a Ph.D. in Computer Engineering and Automation at Politecnico di Milano. He is actually working in both Robotics and Machine Learning, mainly applying, in a practical way, techniques for adaptation and learning to autonomous robotics systems. He is part of the Technical Committee of AI and Robotics conferences, and reviewer for international journals on Robotics and Computational Intelligence. He is the coordinator of the RAWSEEDS project (20062009) an FP6 Specific Support Action to develop a benchmark toolkit for SLAM.