

Vind: a robot self-localization framework

Jon Azpiazu^{1a}, Magnus Bjerkeng¹, Johannes Tjønnås¹ and Esten Ingar Grøtli¹

¹ Department of Applied Cybernetics, SINTEF ICT, Trondheim, Norway

Abstract. In this paper we present a framework for robot localization codenamed *Vind*. The framework allows to configure a multi-sensor setup by describing the configuration and entering the sensor's parameters in a series of text-based and human-readable configuration files. The framework provides, among others, distributed communication capabilities and a state estimation implementation based on the Extended Kalman Filter (EKF). Vind can also be extended to include other state estimation implementations based on clearly defined interfaces and message structures. The aim of the framework is to foster reusability, and provide developers with tools to minimize the effort required to deploy a solution for the self-localization problem. In case of researchers working on the implementation of new state estimate algorithms, it also supports them by providing high level tools for the system integration aspects.

1 Introduction

The self-localization problem is common in any robotic application that requires the robot to navigate through an environment. In order to correctly plan a trajectory and execute the sequence of motions that will take the robot from the current position to the target, the robot must continuously be able to estimate its own position in respect to some known external reference frame [1].

It is very common in robotic applications that one single sensor cannot solve in a satisfactory way the self-localization problem. There are several reasons: the lack of accuracy of the sensor, different coverage areas provided by different sensors, various properties of the sensors that make them unusable under different circumstances or environment types, are some of them. The techniques and theory to combine data originating from different sensor sources is referred to as *sensor fusion* [2] or multi-sensor data fusion. According to [3], the sensors used for solving the robot self-localization problem can be classified in four different groups: odometry and dead-reckoning methods, active beacon systems, landmark navigation and map-based positioning (also known as map matching).

In this paper we present a framework for solving the self-localization problem in mobile robots. The framework is motivated by our own needs, in which the reuse of code in case of a new robot application or a new sensor configuration was a significant challenge, and in most cases led to rewriting major portions of the code. Thus, the framework provides the following set of functionalities, aimed at providing a common structure that can be exploited by various configurations:

- A communication middleware for a distributed system, that allows to transparently distribute the sensors and filter implementations among diverse computing devices.
- Standardized interfaces and message definitions for seamless integration of additional sensors and filters to the framework.
- Sensor fusion algorithms based on recursive Bayesian estimation, including Extended Kalman Filter implementations that supports the integration an arbitrary number of sensors.
- A calibration procedure that minimizes the necessity of manual work.
- Interoperability with external platforms such as Matlab and ROS [4].
- Tools that support debugging problems and configuring the system setup.

The remainder of this paper is organized as follows: Section 2 introduces some alternatives to the Vind framework presented in this paper. Section 3 contains a general description of the Vind framework. Section 4 describes the implementation decisions made to integrate filters based on Bayesian filtering in the framework. Section 5 describes the autocalibration module developed. In Section 6 we use real sensor data to demonstrate the performance of the autocalibration procedure and the Vind framework. Conclusions are given in Section 7.

2 Background and motivation

The main motivation for a self-localization framework is to manage the system's complexity by decoupling the state estimation implementation from

^a Corresponding author: jon.azpiazu@sintef.no

other system integration related problems, such as interfacing with sensors and communication.

There are different robotics middleware alternatives available [5] (ROS, Orocos, Miro, Orca, YARP), and many of them provide their own implementation for robot self-localization. However those middlewares are generic in the sense that they support different aspects of a robotic application, from perception, to low level control and even cognition. The Vind framework in contrast is designed explicitly as a framework for sensor fusion for robot localization. This allows to optimize the system for such application, reducing for example the footprint and dependencies, and setting the requirements for the communication. It can be assumed that ROS is the most relevant system compared to Vind, and in fact the *robot_localization* package [6] pursues very similar objectives. However the communication layer in ROS has several limitations imposed by the original design. This has been recognized by the ROS community, and is reflected on the fact of the big re-design effort that is actually taking place, and that will result with a renewed version of ROS which is codenamed ROS 2.0 or ROS/DDS. An experimental analysis of some of the ROS limitations can be found in [7].

3 The framework architecture

The design of Vind is guided by four main requirements: modularity, scalability, reconfigurability and performance.

- *Modularity*, by being able to transparently divide the different components of the framework among a distributed system, and encapsulating this functionality over clearly defined message definitions and interfaces.
- *Scalability*, as the system can be extended to include support for new sensors as well as implementation of new state estimate algorithms, supporting inheritance within Bayesian filtering framework
- *Reconfigurability*, as the framework makes it easy to adapt the system to different types of robots and different kinds of environments without re-engineering an entire solution. Adaptation to changes in the sensor configuration is dealt with by modifying parameters in a human-readable text configuration file.
- *Performance*, minimizing as much as possible the traffic generated by the framework and the number of dependencies where the framework is able to run.

The communication implementation is built in three different layers. In the messaging layer the ZeroMQ (also spelled as ØMQ or ZMQ) library is used. The ZeroMQ library supports different architectures regarding common patterns found in middlewares or messaging libraries (broker, brokerless, distributed broker, ...). ZeroMQ does not impose any of these models, and allows for implementing any of them according to the use-case requirements. In the case of Vind, it is required to minimize the traffic and latency of the messages. This led to a brokerless implementation, that is communication between the nodes or components of the framework is established in a peer-to-peer fashion. In order to avoid hardcoding where each component runs, which also

worsens the manageability of complex systems, there is a centralized *directory service* that is used to resolve where each of the components is running.

The data layer is implemented through Google protocol buffers. This is an automated mechanism for serializing structured data, with the focus on performance and simplicity. As evaluated by [8], it offers better performance both in the speed for serialization/deserialization, and also generates smaller messages than other commonly used alternatives like JSON or XML. The specification of each message sent throughout the software framework is specified in simple text “.proto” files following a specific proto language.

We then generate data access classes by running a compiler over the message definitions. The resulting access classes provide simple accessors for each field (like name() and set_name()), as well as methods to serialize/parse the whole structure to/from raw bytes.

Even if the use of protocol buffers allow for easy creation of new data types, Vind defines a limited subset of possible messages. This limited number of messages favours interoperability, by making it easier to seamlessly integrate new sensors into the framework as far as the interfaces are respected. Currently Vind specifies six different types of messages, three for the 2D case, and the corresponding three messages extended for robots operating in 3D environments:

- Pose sensors, or subsets of pose, e.g: GPS, camera-based positioning, map matching, radio frequency or ultrasound beacon-positioning systems.
- Twist sensors, or subsets of twist e.g: Doppler radar, optical flow, odometry or rate gyro.
- Acceleration sensors, e.g. accelerometer or angular acceleration sensors.

It is worth noting that each message can optionally include a *mask* field in the message, which allows for per-sensor control of which message fields are used within the state estimate, instead of just reporting high values on the covariance matrix so that the measurement is ignored.

On top of these two layers, there is an additional middleware interface layer. This way, the complexity and specifics of the lower layers is hidden via a unified API. This makes it also possible to create an independence from the communication mechanisms, allowing in the future to change the messaging or data layers with minimum impact for the user.

The configuration of a self-localization solution is based on text files using the YAML format. Using this simple format, a self-localization solution is specified in two parts: first the sensor topology is described, and then the parameters for each sensor are detailed. Any change in the sensor configuration can be reflected by modifying the configuration files.

4 The filter

The implementation of the filter is built around the recursive Bayesian solution, see for instance Chapter 6.2 in [9] and Chapter 2.4 in [10]. The filter can be illustrated through a time update (transformation and diffusion) and

a measurement update (estimation and fusion), see Figure 1. For every time step the a priori state belief distribution is calculated based on the previous posteriori state belief distribution and the control distribution, as shown in:

$$p(x_t|y_{1:t-1}, u_{1:t}) = \int_{\mathbb{R}^n} p(x_t|x_{t-1}, u_t) p(x_{t-1}|y_{1:t-1}, u_{1:t-1}) dx_{t-1} \quad (1)$$

In the measurement update step the posteriori state distribution is calculated by the priori state distribution and the measurement distribution:

$$p(x_t|y_{1:t}, u_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1}, u_{1:t})}{p(y_t|y_{1:t-1}, u_{1:t})} \quad (2)$$

The traditional Kalman Filter (KF) and non-linear variants such as the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF), can be thought of as special cases of the Bayesian recursive filter, each containing a time update and a measurement update. A bank of filters are therefore implemented inheriting the same structure, but with specific implementations of the time and measurement updates.

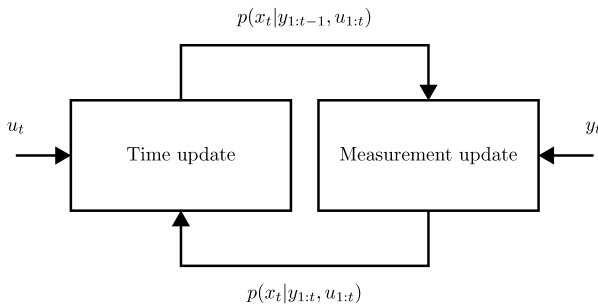


Figure 1. Illustration of Bayes recursive filter

The implementation has focused on fast and easy integration of new sensors, rather than high performance. The framework supports sensor messages in the following categories *pose*, *twist* and *acceleration*. Sensors creating other type of messages can be included by additional filters estimating “measurements” in one of the supported categories.

One of the choices made is that the readings from sensors affecting the integrators furthest down the integrator chain for translational and rotational motion, respectively, are chosen as inputs to the filter, and thus followed by the Bayesian time update step. Other readings are chosen as the model system output and followed by the Bayesian measurement update step. In a navigation system incorporating an Inertial Measurement Unit (IMU), the readings from the gyroscopes and the accelerometers are chosen as inputs. The rationale behind this approach is that the mean of the state estimate is affected directly by these measurements, avoiding delay in information since one would have to wait until the next time update for the measurement to take effect.

Asynchronous measurements are handled by:

- storing the time-stamp given from the measure when the prediction (time update) is required,

- calculating the prediction by previous input measurements,
- and, storing the new measurements for the next prediction step.

An example of a localization system implemented in Vind is given in Figure 2. Here we have chosen to use a Mahony observer [11] to calculate the orientation of the vehicle based on measurements from the IMU.

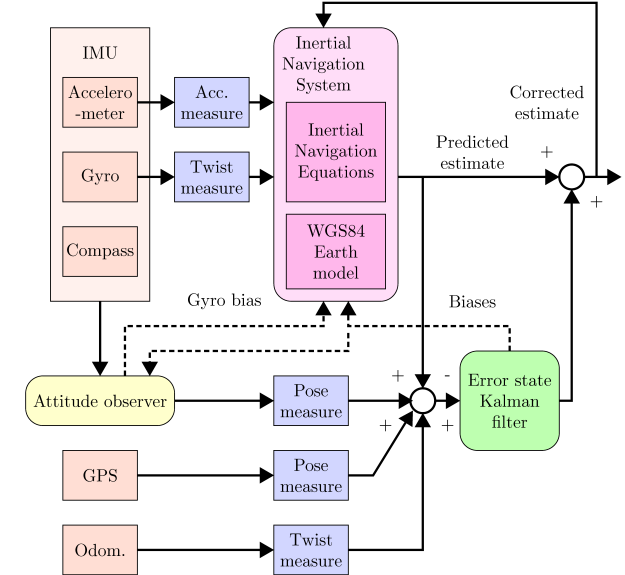


Figure 2. Illustration of a localization system

5 Autocalibration

On robotic platforms with multiple sensors, a necessary integration task is the sensor-to-sensor calibration. For example, on a drone with an IMU and a GPS, one has to identify the transformation between the IMU-frame and the GPS-frame. This calibration is necessary for successful sensor fusion. Moreover, a more accurate calibration results in a more accurate sensor fusion. The purpose of the autocalibration module is twofold:

- To remove the need for time consuming and tedious manual sensor-to-sensor calibration.
- To improve the accuracy of sensor-to-sensor calibration.

The calibration is done by analyzing sensor signals recordings taken from experiments, and inferring the spatial and temporal relationships between them using optimization.

Classical intrasensor calibration methods use Kalman filters to both estimate the state and the calibration constants on-line. They achieve this by including calibration constants in the state, making a linear dynamical model into a non-linear one using either EKF [12] or UKF [13].

The Vind autocalibration module is an extension based on a recent method proposed by [14]. The offline batch processing calibration method proposed by [14] has the following desirable features:

- A maximum likelihood calibration estimate is produced assuming white noise.

- Time-delay calibration is done in the same framework as spatial calibration.
- The size of the optimization problem scales well with data amount.
- The method handles biases, non-constant sampling rates.

The method tries to fit one function to all the sensor data simultaneously. This function is an analytic B-spline which represents the robot pose trajectory as a function of time, see [15]. B-spline functions can be differentiated and time shifted analytically, which makes the objective functions easy to construct. The Levenberg-Marquardt algorithm [16] is used to solve the resulting optimization problem.

The Vind autocalibration extends the implementation in [14], which only performed camera-to-IMU calibration, by supporting multiple sensors in a very generic framework. Sensors providing readings in any of the modalities defined by Vind (pose, twist and acceleration) are supported by the autocalibration procedure.

5.1 Contributions in autocalibration

Some contributions were made to the method described in [14] to generalize the method. This section contains a list of the modifications which were made, and the reasoning behind making them.

5.1.1 Objective weighting

To make each sensor contribute equally to the optimization objective, each sensor objective was weighted by the inverse of the number of samples for that sensor, i.e.

$$g(\mathbf{x}_m, \boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N [x_{m_i} - f(x_{B\text{-spline}}(t_i, \boldsymbol{\theta}))]^2. \quad (3)$$

Here g is the objective function, x_{m_i} is the i 'th measurement for a sensor taken at time t_i , $\boldsymbol{\theta}$ is the vector of calibration constants, N is the number of samples, and f is the function transforming the B-spline parametrized by $x_{B\text{-spline}}$ into the same frame as the measurement data. This choice of weighting makes the contribution of each sensor to the overall objective independent of the number of samples for that vector. Similarly, the continuous bias objective was weighted by $(t_i - t_0)^{-1}$ for each bias, which is the inverse of the duration of the experiment in seconds. This makes the bias objective independent of how long the experiment took to perform.

5.1.2 Validation

In [14] the entire dataset was used in the optimization problem. The Vind autocalibration splits the dataset into an identification and a validation dataset as is typical in system identification, [17]. One is used during optimization, the other is used to check the result after the optimization has terminated. This method is used to avoid

over-parametrization, e.g. by increasing the order of the B-spline too much. Note that this split cannot be done by e.g. splitting the dataset in half at the middle, since the B-spline trajectory output from the optimization only exists for the time interval covered by the identification dataset. Instead, cross-validation is used to overcome this issue, [18]. Figure 3 shows calibration results with and without cross-validation for simulated data. The worse results for the non-validation optimization can be attributed to overestimation. The plot shows a divergence between no validation and cross-validation optimization at 350 model parameters. The (blue) optimization result is decreasing, whereas the quality of the calibration is getting worse. This behaviour indicates that splines with more parameters than 350 variables in this particular dataset are used to fit noise.

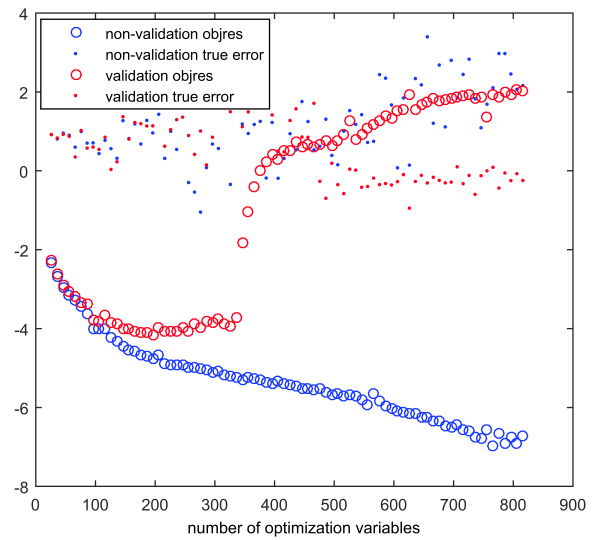


Figure 3. The value of the objective function and the actual calibration error on simulated data, using both cross-validation and no validation.

6 Experiments

We conducted an experiment that allows for different sensor configurations. For the experiment a Pioneer P3-DX robot was used. The robot was equipped with two IMU sensors, a XSens MTi-30 and an ArduIMU v3. In order to provide position data, two sensors were used. First an Hexamite Hx19r was used, which relies on ultrasound for calculating a position (with no orientation). The emitter is fixed to the mobile robot, and four receivers are mounted in the lab's ceiling. The emitter calculates the time-of-flight for the signal from the receivers, and then the position is calculated by trilateration. The other sensor providing position is an OptiTrack motion capture system. This is a high precision optical system that uses infrared markers fixed to the mobile robot, and a series of cameras that triangulate the markers' positions to calculate a full pose.

6.1 Calibration

Several experiments were conducted to test the autocalibration module. The sensors mentioned in the previous section were mounted on the Pioneer mobile robot. This was done to show that Vind handles rapid deployment and redeployment of sensors. In Figure 4, the raw Hexamite and OptiTrack data is shown, alongside the calibrated data. The IMU-calibration is not shown. The experiment duration was approximately 30 seconds, and the optimization routine terminated in approximately 2 minutes.

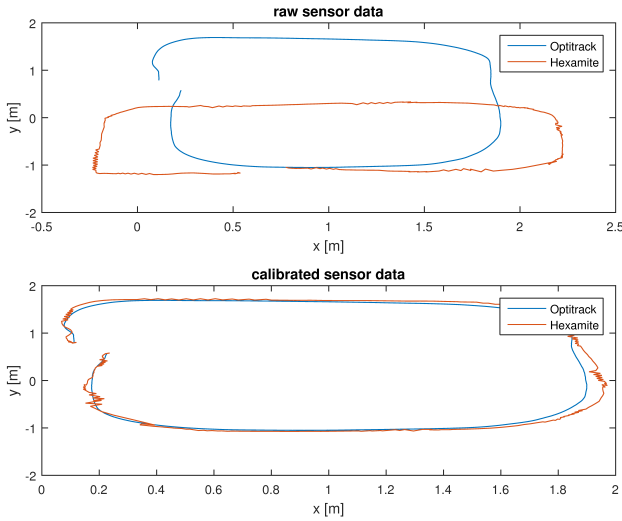


Figure 4. Calibration of the Hexamite data.

6.2 Filtering

In order to illustrate the framework ease of reconfiguration, a comparison of filter solution with different sensor combinations is considered. The measurements from the OptiTrack motion capture system is treated as the ground truth and compared with the other filter solutions through a *Mean Squared Error* score, as shown in Figure 5. In the different estimation solution the *OptiTrack_{Noise}* sensor data is constructed by the *OptiTrack* position measurements with the added noise $\sigma = 0.5m$. The system model and the sensor expected noises used in the filter are given in Table 1.

Table 1. Filter tuning parameters

Sensor variance	Measure type
$R_{OptiTrack_{Noise}} = \text{diag}([1,1,1])$	x and y position, heading angle
$R_{Hexamite} = \text{diag}([1,1]) * 10^{-2}$	x and y position
$Q_{Xsens} = \text{diag}([1,1,1,1,1,1,1,1]) * 10^{-2}$	process model: $[\dot{x}, \dot{y}, \dot{x}, \dot{y}, \psi, b_{accx}, b_{accy}, b_r]$
$Q_{ArduIMU} = \text{diag}([1, 1]*10^{-2}, [1, 1], 1, [1, 1], 1) * 10^{-2}$	process model: $[\dot{x}, \dot{y}, \dot{x}, \dot{y}, \psi, b_{accx}, b_{accy}, b_r]$

Any sensor combination which results in an observable estimation system may be considered. Sensor

fallouts are handled directly in the framework, but as long duration without pose information will render the system locally in time unobservable, the solution will drift and the system will have poor but known estimation performance.

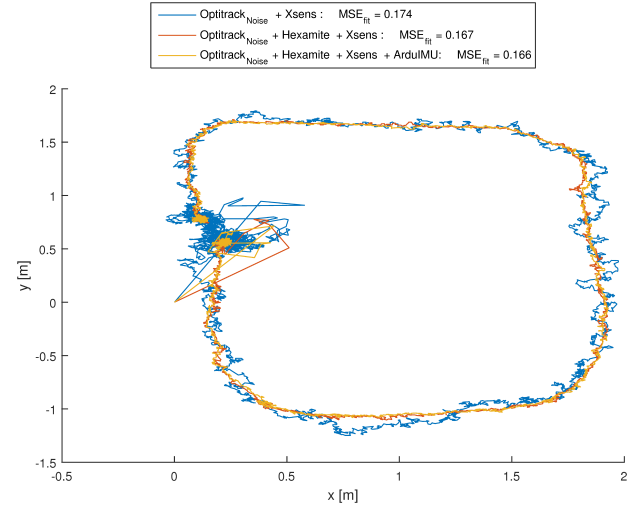


Figure 5. Position estimates given different sensor-sensor combination in the filtering framework.

7 Conclusions and future work

In this paper we have presented the Vind framework for robot localization. The framework provides distributed communication capabilities, a state estimation implementation based on EKF, an autocalibration module and the ability for easy reconfiguring a localization solution based on plain text configuration files. The framework can also be easily extended to include additional sensors and additional sensor fusion algorithm implementations. We have demonstrated the use of the framework in a typical scenario for a mobile robot.

Future plans for Vind include implementing and integrating further sensor fusion algorithms. Moreover we are considering tighter integration algorithms, where lower level sensor data is fused and coupled together with incomplete data originated by other sensors. The objective would be then to use sensor data that could not be used in a standard framework, for instance, when a triangulation-based sensor is not able to triangulate because it is not receiving the signal for a sufficient number of beacons, but the data from those beacons could be combined with other sensors to improve the results.

Acknowledgments

This work was partially supported by European Research Programme ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems), project R5-COP (Reconfigurable ROS-based Resilient Reasoning Robotic Co-operating Systems) and NextGenIMR (Next Generation subsea inspection, maintenance and repair operations).

The authors would like to thank Sigurd A. Fjerdigen for his participation in the design and development of the framework.

References

1. J. Leonard, H. Durrant-Whyte, *IEEE Transactions on Robotics and Automation* **7**, 376 (1991)
2. J. Gu, M. Meng, A. Cook, P. Liu, *Sensor fusion in mobile robot: some perspectives*, in *Proc. of the 4th World Congress on Intelligent Control and Automation* **2**, (2002), pp. 1194–1199, ISBN 0-7803-7268-9
3. J. Borenstein, H. Everett, L. Feng, Tech. rep., University of Michigan (1996)
4. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, *ROS: an open-source Robot Operating System*, in *ICRA Workshop on Open Source Software* (2009)
5. A. Elkady, T. Sobh, *Journal of Robotics* (2012)
6. T. Moore, D. Stouch, *A Generalized Extended Kalman Filter Implementation for the Robot Operating System*, in *Proc. of the 13th Int. Conf. on Intelligent Autonomous Systems* (Springer, 2014)
7. A. Shakhimardanov, N. Hochgeschwender, M. Reckhaus, G.K. Kraetzschmar, *Analysis of software connectors in robotics*, in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2011), pp. 1030–1035
8. A. Sumaray, S. Makki, *A comparison of data serialization formats for optimal efficiency on a mobile platform*, in *Proc. of the 6th Int. conf. on Ubiquitous Information Management and Communication* (2012)
9. F. Gustafsson, *Statistical Sensor Fusion* (Studentlitteratur, 2012)
10. S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics* (The MIT Press, 2006)
11. R. Mahony, T. Hamel, J.M. Pflimlin, *IEEE Transactions on Automatic Control* **53**, 1203 (2008)
12. F.M. Mirzaei, S. Roumeliotis et al., *IEEE Transactions on Robotics* **24**, 1143 (2008)
13. T. Beravs, J. Podobnik, M. Munih, *Instrumentation and Measurement*, *IEEE Transactions on* **61**, 2501 (2012)
14. P. Furgale, J. Rehder, R. Siegwart, *Unified temporal and spatial calibration for multi-sensor systems*, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2013), pp. 1280–1286, ISBN 978-1-4673-6358-7, ISSN 2153-0858
15. J.C. Beatty, B.A. Barsky, *An introduction to splines for use in computer graphics and geometric modelling* (Morgan Kaufmann, 1995)
16. J. Nocedal, S. Wright, *Numerical optimization* (Springer Science & Business Media, 2006)
17. L. Ljung, Englewood Cliffs (1987)
18. R.R. Picard, D. Cook, *Journal of the American Statistical Association* **79**, 575 (1984)

Authors' background

Your Name	Title*	Research Field	Personal website
Jon Azpiazu	Researcher	Robotics, Computer Vision	https://www.sintef.no/en/information-and-communication-technology-ict/departments/applied-cybernetics/
Magnus Bjerkeng	Researcher	Robotics, control and estimation	https://www.sintef.no/en/information-and-communication-technology-ict/departments/applied-cybernetics/
Johannes Tjønnås	Researcher	Robotics, control and estimation	https://www.sintef.no/en/information-and-communication-technology-ict/departments/applied-cybernetics/
Esten Ingar Grøtli	Researcher	Robotics, control and estimation	https://www.sintef.no/en/information-and-communication-technology-ict/departments/applied-cybernetics/

*This form helps us to understand your paper better, **the form itself will not be published.**

*Title can be chosen from: master student, Phd candidate, assistant professor, lecture, senior lecture, associate professor, full professor