A Framework for Simultaneous Localization and Mapping Utilizing Model Structure

Thomas B. Schön, Rickard Karlsson, David Törnqvist, Fredrik Gustafsson

Division of Automatic Control
E-mail: schon@isy.liu.se, rickard@isy.liu.se, tornqvist@isy.liu.se, fredrik@isy.liu.se

11th May 2007

Report no.: LiTH-ISY-R-2782
Accepted for publication in the 10th International Conference on Information Fusion (FUSION 2007)

Address:
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

WWW: http://www.control.isy.liu.se
Abstract
The basic nonlinear filtering problem for dynamical systems is considered. Approximating the optimal filter estimate by particle filter methods has become perhaps the most common and useful method in recent years. Many variants of particle filters have been suggested, and there is an extensive literature on the theoretical aspects of the quality of the approximation. Still, a clear cut result that the approximate solution, for unbounded functions, converges to the true optimal estimate as the number of particles tends to infinity seems to be lacking. It is the purpose of this contribution to give such a basic convergence result.

Keywords: Nonlinear filters, particle filter, convergence, dynamic systems.
A Framework for Simultaneous Localization and Mapping Utilizing Model Structure

Thomas B. Schön, Rickard Karlsson, David Törnqvist, and Fredrik Gustafsson
Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE–581 83 Linköping, Sweden
Email: {schon, rickard, tornqvist, fredrik}@isy.liu.se

Abstract—This contribution aims at unifying two recent trends in applied particle filtering (PF). The first trend is the major impact in simultaneous localization and mapping (SLAM) applications, utilizing the FastSLAM algorithm. The second one is the implications of the marginalized particle filter (MPF) or the Rao-Blackwellized particle filter (RBPF) in positioning and tracking applications. An algorithm is introduced, which merges FastSLAM and MPF, and the result is an MPF algorithm for SLAM applications, where state vectors of higher dimensions can be used. Results using experimental data from a 3D SLAM development environment, fusing measurements from inertial sensors (accelerometer and gyro) and vision are presented.

Keywords: Rao-Blackwellized/marginalized particle filter, sensor fusion, simultaneous localization and mapping, inertial sensors, vision.

I. INTRODUCTION

The main task in localization/positioning and tracking is to estimate, for instance, the position and orientation of the object under consideration. The particle filter (PF), [1], [2], has proven to be an enabling technology for many applications of this kind, in particular when the observations are complicated nonlinear functions of the position and heading [3]. Furthermore, the Rao-Blackwellized particle filter (RBPF) also denoted the marginalized particle filter (MPF), [4]–[9] enables estimation of velocity, acceleration, and sensor error models by utilizing any linear Gaussian sub-structure in the model, which is fundamental for performance in applications as surveyed in [10]. As described in [9], the MPF splits the state vector $x_t$ into two parts, one part $x^p_t$ which is estimated using the particle filter and another part $x^k_t$ where Kalman filters are applied. Basically, it uses the following factorization of the posterior distribution of the state vector, which follows from Bayes’ rule,

$$p(x_{1:t}, x^k_t | y_{1:t}) = p(x^k_t|x_{1:t}, y_{1:t})p(x_{1:t} | y_{1:t}),$$

(1)

where $y_{1:t} = \{y_1, \ldots, y_t\}$ denotes the measurements up to time $t$. If the model is conditionally linear Gaussian, i.e., if the term $p(x^k_t|x_{1:t}, y_{1:t})$ is linear Gaussian, it can be optimally estimated using the Kalman filter, whereas for the second factor we have to resort to the PF.

Simultaneous localization and mapping (SLAM) is an extension of the localization or positioning problem to the case where the environment is un-modeled and has to be mapped on-line. An introduction to the SLAM problem is given in the survey papers [11], [12] and the recent book [13]. From a sensor point of view, there are two ways of tackling this problem. The first way is to use only one sensor, such as vision, see e.g., [14]–[17] and the second way is to fuse measurements from several sensors. This work considers the latter.

The FastSLAM algorithm introduced in [18] has proven to be an enabling technology for such applications. FastSLAM can be seen as a special case of RBPF/MPF, where the map state $m_t$, containing the positions for all landmarks used in the mapping, can be interpreted as a linear Gaussian state. The main difference is that the map vector is a constant parameter with a dimension increasing over time, rather than a time-varying state with a dynamic evolution over time. The derivation is completely analogous to (1), and makes use of the following factorization

$$p(x_{1:t}, m_t | y_{1:t}) = p(m_t | x_{1:t}, y_{1:t})p(x_{1:t} | y_{1:t}).$$

(2)

The FastSLAM algorithm was originally devised to solve the SLAM problem for mobile robots, where the dimension of the state vector is small, typically consisting of three states (2D position and a heading angle) [13]. This implies that all platform states can be estimated by the PF.

Paralleling the evolution of PF applications to high dimensional state vectors, the aim of this contribution is to unify the ideas presented in [9], [19] in order to extend the FastSLAM [18] algorithm to be able to cope with high dimensional state vectors as well. Basically, the main result follows from

$$p(x^p_{1:t}, x^k_t, m_t | y_{1:t})$$

$$= p(m_t | x^k_t, x^p_{1:t}, y_{1:t})p(x^k_t | x^p_{1:t}, y_{1:t})p(x^p_{1:t} | y_{1:t}).$$

(3)

The derived algorithm is applied to experimental data from a development environment tailored to provide accurate values of ground truth. Here, a high precision industrial robot is programmed to move, possibly using its 6 degrees-of-freedom...
The key factorization, which allows us to solve this problem successfully is

\[
p(x_{1:t},x^k_t, m_t | y_{1:t}) = \prod_{j=1}^{M_t} p(m_{j,t} | x_{1:t}, x^k_t, y_{1:t}) p(x^k_t | x_{1:t}, y_{1:t}) p(x_{1:t} | y_{1:t})
\]

(extended) Kalman filter

In order to devise an estimator for (6) a system model and a measurement model are needed. The former describes the dynamic behaviour of the platform, that is how the state \( x_t \) evolves over time. The measurement model describes the sensors, i.e., it consists of equations relating the measurements \( y_t \) to the state \( x_t \). We want a general algorithm, which is applicable to many different platforms (aircraft, helicopters, cars, etc.). Hence, the model structure should be as general as possible, leading us to,

\[
x^p_{t+1} = f_p^t(x^p_t) + A^p_v(x^p_t)x^k_t + G^p_t(x^p_t)w^p_t, \quad (8a)
\]

\[
x^k_{t+1} = f^k_t(x^k_t) + A^k_t(x^k_t)x^r_t + G^k_t(x^k_t)w^k_t, \quad (8b)
\]

\[
m_{j,t+1} = m_{j,t}, \quad (8c)
\]

\[
y_{1,t} = h_{1,t}(x^p_t) + C_t(x^p_t)x^k_t + e_{1,t}, \quad (8d)
\]

\[
y_{2,t} = h_{2,t}(x^p_t) + H_{j,t}(x^p_t)m_{j,t} + e_{2,t}, \quad (8e)
\]

where \( j = 1, \ldots, M_t \) and the noise for the platform states is assumed white and Gaussian distributed with

\[
w_t = \begin{pmatrix} w^p_t \\ w^k_t \end{pmatrix} \sim \mathcal{N}(0, Q_t), \quad Q_t = \begin{pmatrix} Q^p_t & Q^{pk}_{t} \\ (Q^{pk})^T & Q^k_t \end{pmatrix}.
\]

To simplify the notation in the rest of the paper, denote \( f_p^t(x^p_t) \) with \( f^p_t \), \( A^p_v(x^p_t) \) with \( A^p_v \) and so on. The measurement noise is assumed white and Gaussian distributed according to

\[
e_{1,t} \sim \mathcal{N}(0, R_{1,t}), \quad (8g)
\]

\[
e^{(j)}_{2,t} \sim \mathcal{N}(0, R_{2,t}), \quad j = 1, \ldots, M_t. \quad (8h)
\]

Finally, \( x^0_0 \) is Gaussian,

\[
x^0_0 \sim \mathcal{N}(\bar{x}_0, \bar{P}_0), \quad (8i)
\]

and the density for \( x^p_0 \) can be arbitrary, but it is assumed known.

There are two different measurement models, (8d) and (8e), where the former only measures quantities related to the platform, whereas the latter will also involve the map states. Section IV describes a detailed application example using experimental data, where (8d) is used to model inertial sensors and (8e) is used to model a camera.

III. PARTICLE FILTER FOR SLAM UTILIZING STRUCTURE

In Section III-A the proposed SLAM algorithm is given and in the subsequent sections the details of the algorithm are discussed.
A. Algorithm

The algorithm presented in this paper draws on several rather well known algorithms. It is based on the RBPF/MPF method, [4]–[9]. The FastSLAM algorithm [18] is extended by not only including the map states, but also the states corresponding to a linear Gaussian sub-structure present in the model for the platform. Assuming that the platform is modeled in the form given in (8), the SLAM-method utilizing structure is given in Algorithm 1.

**Algorithm 1: Particle filter for SLAM utilizing structure**

1) Initialize the particles

\[ x_{1|0}^{p,(i)} \sim p(x_{1|0}^{p}), \]
\[ x_{1|0}^{k,(i)} = x_{1|0}^{k}, \]
\[ p_{1|0}^{k,(i)} = \bar{p}_{1|0}, \quad i = 1, \ldots, N, \]

where \( N \) denotes the number of particles.

2) If there is a new map related measurement available, perform data association for each particle, otherwise proceed to step 3.

3) Compute the importance weights according to

\[ \gamma_{t}^{(i)} = p(y_{t}|x_{t}^{p,(i)}, y_{t-1}), \quad i = 1, \ldots, N, \]

and normalize \( \bar{\gamma}_{t}^{(i)} = \gamma_{t}^{(i)}/\sum_{j=1}^{N} \gamma_{t}^{(j)} \).

4) Draw \( N \) new particles with replacement (resampling) according to, for each \( i = 1, \ldots, N \)

\[ \text{Pr}(x_{t+1|t}^{p,(i)} = x_{t+1|t}^{p,(j)}) = \bar{\gamma}_{t}^{(i)}, \quad j = 1, \ldots, N. \]

5) If there is a new map related measurement, perform map estimation and management (detailed below), otherwise proceed to step 6.

6) Particle filter prediction and Kalman filter (for each particle \( i = 1, \ldots, N \))

a) Kalman filter measurement update,

\[ p(x_{t}^{k}|x_{t+1}^{p}, y_{1:t}) = N(\hat{x}_{t+1|t}^{k,(i)}, P_{t+1|t}^{k,(i)}), \]

where \( \hat{x}_{t+1|t}^{k,(i)} \) and \( P_{t+1|t}^{k,(i)} \) are given in (11).

b) Time update for the nonlinear particles,

\[ x_{t+1|t}^{p,(i)} \sim p(x_{t+1|t}^{p,(i)}|x_{t+1|t}^{p,(i)}, y_{1:t}). \]

c) Kalman filter time update,

\[ p(x_{t}^{k}|x_{t+1|t}^{p}, y_{1:t}) = N(\hat{x}_{t+1|t}^{k,(i)}, P_{t+1|t}^{k,(i)}), \]

where \( \hat{x}_{t+1|t}^{k,(i)} \) and \( P_{t+1|t}^{k,(i)} \) are given by (12).

7) Set \( t := t+1 \) and iterate from step 2.

Note that \( y_{t} \) denotes the measurements present at time \( t \). The following theorem will give all the details for how to compute the Kalman filtering quantities. It is important to stress that all embellishments available for the particle filter can be used together with Algorithm 1. To give one example, the so-called FastSLAM 2.0 makes use of an improved proposal distribution in step 6b [20].

**Theorem 1:** Using the model given by (8), the conditional probability density functions for \( x_{t}^{k} \) and \( x_{t+1}^{k} \) are given by

\[ p(x_{t}^{k}|x_{t+1}^{p}, y_{1:t}) = N(\hat{x}_{t+1|t}^{k,(i)}, P_{t+1|t}^{k,(i)}), \]

where

\[ \hat{x}_{t+1|t}^{k} = \hat{x}_{t+1|t}^{k} + A_{t}^{k}(Q_{t}^{k})^{T}(G_{t}^{k}Q_{t}^{k})^{-1} z_{t} \]

\[ + f_{t}^{k} + L_{t}(z_{t} - A_{t}^{k}\hat{x}_{t+1|t}^{k}), \]

\[ P_{t+1|t}^{k} = A_{t}^{k}P_{t+1|t}^{k}(A_{t}^{k})^{T} + G_{t}^{k}(Q_{t}^{k})^{T} - L_{t}S_{2,t}L_{t}^{T}, \]

\[ S_{2,t} = A_{t}^{k}P_{t+1|t}^{k}(A_{t}^{k})^{T} + G_{t}^{k}(Q_{t}^{k})^{T}, \]

\[ L_{t} = A_{t}^{k}P_{t+1|t}^{k}(A_{t}^{k})^{T}S_{2,t}^{-1}, \]

and

\[ z_{t} = x_{t+1}^{p} - f_{t}^{p}, \]

\[ \hat{X}_{t}^{k} = A_{t}^{k} - G_{t}^{k}(Q_{t}^{k})^{T}(Q_{t}^{k}Q_{t}^{k})^{-1} A_{t}^{p}, \]

\[ Q_{t}^{k} = Q_{t}^{k} - (Q_{t}^{k})^{T}(Q_{t}^{k}Q_{t}^{k})^{-1} Q_{t}^{k}. \]

**Proof:** See [9].

B. Data Association

Data association is a complicated problem, but it has been studied extensively in the literature for many tracking applications, see e.g., [21]–[23]. Classical methods such as the nearest neighbor (NN), probabilistic data association (PDA), joint probabilistic data association (JPDA) or multi-hypothesis tracking (MHT) exist for single and multiple targets. Depending on the number of targets, the false alarm rate and the probability of detection, these methods varies in performance and ability to express these phenomena. The methods mentioned above were originally developed to be used together with estimators based on Kalman filters. The use of particle filters opens up for other data association methods, typically more integrated with the filter.

The particle implementation of the SLAM problem will lead to an increased complexity for the data association. This is because for each particle in the filter there exist several map entries. Hence, many classical association methods will be too computationally intensive for a direct implementation.
C. Likelihood Computation

In order to compute the importance weights \( \{\gamma_t^{(i)}\}_{i=1}^N \) in Algorithm 1, the following likelihoods have to be evaluated
\[
\gamma_t^{(i)} = p(y_t|x^{(i)}_{1:t}, y_{1:t-1}), \quad i = 1, \ldots, N.
\]  
(14)
The standard way of performing this type of computation is simply to marginalize the Kalman filter variables \( x^k_t \) and \( \{m_{j,t}\}_{j=1}^M \)
\[
p(y_t|x^{(i)}_{1:t}, y_{1:t-1}) = \int p(y_t, x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) dx^k_t dm_{i|x^{(i)}_{1:t}}, \quad i = 1, \ldots, N.
\]  
(15)
where
\[
p(y_t, x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) = \int p(y_t|x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) \times p(x^k_t|m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) \text{d}x^k_t \text{d}m_{i|x^{(i)}_{1:t}},
\]  
(16)
Let us consider the case where both \( y_{1:t} \) and \( y_{2:t} \) are present, i.e., \( y_t = (y^T_{1:t}, y^T_{2:t})^T \). Note that the cases where either \( y_{1:t} \) or \( y_{2:t} \) are present are obviously special cases. First of all, the measurements are conditionally independent given the state, implying that
\[
p(y_t|x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) = p(y_t|x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) \prod_{j=1}^M p(y_j^{(j)}|x^k_t, m_{j|x^{(i)}_{1:t}}, y_{1:t-1}),
\]  
(17)
Now, inserting (17) into (16) gives
\[
p(y_t, x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) = \int p(y_t|x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) \times \prod_{j=1}^M p(\{m_{j|x^{(i)}_{1:t}}, y_{1:t-1}\}) \text{d}x^k_t \text{d}m_{i|x^{(i)}_{1:t}},
\]  
(18)
which inserted in (15) finally results in
\[
p(y_t|x^{(i)}_{1:t}, y_{1:t-1}) = \int p(y_t|x^k_t, m_{i|x^{(i)}_{1:t}}, y_{1:t-1}) \times \prod_{j=1}^M \int p(y_j^{(j)}|x^k_t, m_{j|x^{(i)}_{1:t}}, y_{1:t-1}) \text{d}x^k_t \text{d}m_{j|x^{(i)}_{1:t}},
\]  
(19)
All the densities present in (19) are known according to
\[
p(x^k_t|x^{(i)}_{1:t-1}, y_{1:t-1}) = \mathcal{N}(x^k_t|\tilde{x}^k_{1:t-1}, P_{k|t-1}),
\]  
(20a)
\[
p(m_{j,t}|x^{(i)}_{1:t-1}, y_{1:t-1}) = \mathcal{N}(m_{j,t}|\tilde{m}_{j,t-1}, \Sigma_{j,t-1}),
\]  
(20b)
\[
p(y_{1:t}|x^k_t, m_{i|x^{(i)}_{1:t}}) = \mathcal{N}(y_{1:t}|h_{1:t} + Ox^k_t, R_1),
\]  
(20c)
\[
p(y_{2:t}|x^k_t, m_{j|x^{(i)}_{1:t}}) = \mathcal{N}(y_{2:t}|h_{2:t} + H_{j,t}m_{j,t}, R_2).
\]  
(20d)
Here it is important to note that the standard FasSLAM approximation has been invoked in order to obtain (20d). That is, the measurement equation (8e) is linearized with respect to the map states \( m_{j,t} \). The reason for this approximation is that we for computational reasons want to use a model suitable for the RBPF/MPF, otherwise the dimension will be much too large for the particle filter to handle. Using (20), the integrals in (19) can now be solved, resulting in
\[
p(y_t|x^{(i)}_{1:t}, y_{1:t-1}) = \mathcal{N}(y_{1:t} - h_{1,t} - C_t\tilde{x}^{k,(i)}_{t-1}, C_tP^{(i)}_{t-1}C^T_t) \times \prod_{j=1}^M \mathcal{N}(y_j^{(j)} - h_{j,t}\tilde{m}_{j,t-1} - H_{j,t}\Sigma_{j,t-1}(H_{j,t})^T + R_2).
\]  
(21)

D. Map Estimation and Map Management

A simple map consists of a collection of map point entries \( \{m_{j,t}\}_{j=1}^M \), each consisting of:
\begin{itemize}
  \item \( \tilde{m}_{j,t} \) – estimate of the position (three dimensions).
  \item \( \Sigma_{j,t} \) – covariance for the position estimate.
\end{itemize}
Note that this is a very simple map parameterization. Each particle has an entire map estimate associated to it. Step 5 of Algorithm 1 consists of updating these map estimates in accordance with the new map-related measurements that are available. First of all, if a measurement has been successfully associated to a certain map entry, it is updated using the standard Kalman filter measurement update according to
\[
m_{j,t} = m_{j,t-1} + K_{j,t}(y_{2,t}^j - h_{2,t}),
\]  
(22a)
\[
\Sigma_{j,t} = (I - K_{j,t}H_{j,t}^T)\Sigma_{j,t-1},
\]  
(22b)
\[
K_{j,t} = \Sigma_{j,t-1}H_{j,t}(H_{j,t}\Sigma_{j,t-1}H_{j,t}^T + R_2)^{-1}.
\]  
(22c)
If an existing map entry is not observed, the corresponding map estimate is simply propagated according to its dynamics, i.e., it is unchanged
\[
m_{j,t} = m_{j,t-1},
\]  
(23a)
\[
\Sigma_{j,t} = \Sigma_{j,t-1}.
\]  
(23b)
It has to be checked if any of the entries should be removed from the map.
Finally, initialization of new map entries have to be handled. If \( h_{2,t}(x^p_{m_{j,t}}) \) is bijective with respect to the map \( m_{j,t} \) this can be used to directly initialize the position from the measurement \( y_{2,t} \). However, this is typically not the case, implying that there is a need for more than one measurement in order to be able to initialize a new map entry, utilizing for example triangulation.

E. Approximate Algorithm

The computational complexity of Algorithm 1 will inevitably be high, implying that it is interesting to consider ideas on how to reduce it. In the present context multirate sensors (sensors providing measurements with different sampling times) are typically used. This can be exploited in a way similar to that in [19]. The idea is that, rather than running Algorithm 1 at the same frequency as the fast sensor, it is executed at the same frequency as the slow sensor, with
a nested filter handling the fast sensor measurements. This approach involves approximations.

In this way both the linear Gaussian sub-structure in the platform model and the multi-rate properties of the sensors are exploited in order to obtain an efficient algorithm [19].

IV. APPLICATION EXAMPLE

In this section we provide a rather detailed treatment of a SLAM application, where Algorithm 1 is used to fuse measurements from a camera, three accelerometers and three gyroscopes. The sensor has been attached to a high precision 6 DoF ABB IRB1440 industrial robot. The reason for this is that the robot will allow us to make repeatable measurements from a camera, three accelerometers and three gyroscopes. The sensor has been attached to a high precision camera, three accelerometers and three gyroscopes. The sensor has been attached to a high precision camera, three accelerometers and three gyroscopes.

The main objective is to find the position and orientation of the sensor from sensor data only, despite problems such as biases in the measurements. In the surrounding neighbourhoid features are placed in such a way that the vision system can easily extract features. These are used in SLAM to reduce the problem caused by inertial drift and bias in the IMU sensor.

A. Model

The basic part of the state vector consists of position \( p_t \in \mathbb{R}^3 \), velocity \( v_t \in \mathbb{R}^3 \), and acceleration \( a_t \in \mathbb{R}^3 \), all described in an earth-fixed reference frame. Furthermore, the state vector is extended with bias states for acceleration \( b_{a,t} \in \mathbb{R}^3 \), and angular velocity \( b_{\omega,t} \in \mathbb{R}^3 \) in order to account for sensor imperfections. The state vector also contains the angular velocity \( \omega_t \) and a unit quaternion \( q_t \), which is used to parameterize the orientation.

In order to put the model in the RBPF/MPF framework, the state vector is split into two parts, one estimated using Kalman filters \( x^k_t \) and one estimated using the particle filter \( x^p_t \). Hence, define

\[
\begin{align*}
    x^k_t &= \begin{pmatrix} v_t^T & a_t^T \end{pmatrix}^T \begin{pmatrix} b_{\omega,t}^T & \omega_t^T \end{pmatrix}^T, \quad (24a) \\
    x^p_t &= \begin{pmatrix} p_t^T & q_t^T \end{pmatrix}^T, \quad (24b)
\end{align*}
\]

which means \( x^k_t \in \mathbb{R}^{15} \) and \( x^p_t \in \mathbb{R}^7 \). In inertial estimation it is essential to clearly state which coordinate system any entity is expressed in. Here the notation is simplified by suppressing the coordinate system for the earth-fixed states, which means that

\[
\begin{align*}
    p_t &= p_t^e, \\
    v_t &= v_t^e, \\
    a_t &= a_t^e, \\
    w_t &= w_t^e, \\
    b_{\omega,t} &= b_{\omega,t}^e, \\
    b_{a,t} &= b_{a,t}^e.
\end{align*} \quad (25a)
\]

Likewise, the unit quaternions represent the rotation from the earth-fixed system to the body (IMU) system, since the IMU is rigidly attached to the body (strap-down),

\[
q_t = q_t^{be} = \begin{pmatrix} q_{t,0} & q_{t,1} & q_{t,2} & q_{t,3} \end{pmatrix}^T. \quad (26)
\]

The quaternion estimates are normalized, to make sure that they still parameterizes an orientation. Further details regarding orientation and coordinate systems are given in Appendix A.

1) Dynamic Model: The dynamic model describes how the platform and the map evolve over time. These equations are given below, in the form (8a) – (8d), suitable for direct use in Algorithm 1.

\[
\begin{align*}
    \begin{pmatrix}
        v_{t+1} \\
        a_{t+1} \\
        b_{\omega,t+1} \\
        b_{a,t+1} \\
        \omega_{t+1}
    \end{pmatrix} &= \begin{pmatrix}
        I & TI & 0 & 0 & 0 \\
        0 & I & 0 & 0 & 0 \\
        0 & 0 & I & 0 & 0 \\
        0 & 0 & 0 & I & 0 \\
        0 & 0 & 0 & 0 & I
    \end{pmatrix}
    \begin{pmatrix}
        v_t \\
        a_t \\
        b_{\omega,t} \\
        b_{a,t} \\
        \omega_t
    \end{pmatrix} \\
    &+ \begin{pmatrix}
        0 & 0 & 0 & 0 \\
        I & 0 & 0 & 0 \\
        0 & I & 0 & 0 \\
        0 & 0 & I & 0 \\
        0 & 0 & 0 & 1
    \end{pmatrix}
    \begin{pmatrix}
        w_{1,t} \\
        w_{2,t} \\
        w_{3,t} \\
        w_{4,t}
    \end{pmatrix}
    + u^p_t, \quad (27a)
\end{align*}
\]

\[
\begin{align*}
    \begin{pmatrix}
        p_{t+1} \\
        q_{t+1}
    \end{pmatrix} &= \begin{pmatrix}
        p_t & q_t
    \end{pmatrix}
    + \begin{pmatrix}
        TI & T^2 I & 0 \times 9 \\
        0_4 & 0_4 & -T \times S(q)
    \end{pmatrix}
    \begin{pmatrix}
        v_t \\
        a_t \\
        b_{\omega,t} \\
        b_{a,t} \\
        \omega_t
    \end{pmatrix}
    + u^p_t, \quad (27b)
\end{align*}
\]

where

\[
\tilde{S}(q) = \begin{pmatrix}
    -q_1 & -q_2 & -q_3 \\
    q_0 & q_3 & -q_2 \\
    -q_3 & q_0 & q_1 \\
    q_2 & -q_1 & q_0
\end{pmatrix}, \quad (28)
\]

and where \( I \) denotes the \( 3 \times 3 \) unit matrix and \( 0 \) denotes the \( 3 \times 3 \) zero matrix, unless otherwise stated. The process noise \( u^p_t \) is assumed to be independent and Gaussian, with covariance \( Q_t^p = \text{diag}(Q_{a_t}, Q_{b_{\omega,t}}, Q_{b_{a,t}}, Q_{\omega}) \).

2) Measurement Model – Inertial Sensors: The IMU consists of accelerometers measuring accelerations \( y_{a,t} \) in all three dimensions, a gyroscope measuring angular velocities \( y_{\omega,t} \) in three dimensions and a magnetometer measuring the direction to the magnetic north pole. Due to the fairly magnetic environment it is just the accelerometers and gyroscopes that are used for positioning. The inertial sensors operate at 100 Hz. For further details on inertial sensors, see for instance [24]–[26]. The inertial measurements are related to the states according
to,
\[
y_{1,t} = \begin{pmatrix} y_{w,t} \\ y_{n,t} \end{pmatrix} = \begin{pmatrix} 0 \\ -R(q_t)g^c \end{pmatrix} h(x_t^c) + \begin{pmatrix} v_t \\ a_t \\ b_{\omega,t} \\ b_{\nu,t} \\ \omega_t \\ e_t \end{pmatrix},
\]

which obviously is in the form required by (8). The measurement noise \( e_t \) is assumed Gaussian with covariance \( R_t = \text{diag}(R_w, R_a) \).

3) Measurement Model – Camera: Before the camera images are used they are adjusted according to the calibration. This allows us to model the camera using the pinhole model with focal length \( f \), according to [27], [28],
\[
y_{2,t} = y_{m,t} = \frac{1}{z_t} \begin{pmatrix} x_t^c \\ y_t^c \end{pmatrix} + e_{3,t},
\]

where
\[
m_{j,t}^c = \begin{pmatrix} x_t^c \\ y_t^c \end{pmatrix} = R(q_t^{eb})R(q_t^{be})(m_{j,t} - p_t) + r^c.
\]

Here, \( r^c \) is a fixed vector representing the translation between the camera and the IMU and \( q_t^{eb} \) is the unit quaternion describing the rotation from the IMU to the camera. The covariance for the measurement noise is denoted \( R_c \).

This particular sensor is equipped with an internal camera, which is synchronized in time with the inertial measurements. This provides a good setup for fusing vision information with the inertial information. Images are available at 12.5 Hz in a resolution of 340 x 280 pixels. In order to use vision for feature extraction and estimation we have made use of standard camera calibration techniques, see e.g., [29].

The features are not exactly in the form suitable for marginalization in the particle filter. Hence, we are forced to use an approximation in order to obtain a practical algorithm. The standard approximation [13] is in this case simply to linearize the camera measurement equations according to,
\[
\begin{align*}
&y_{m,j, t} = h^c(m_{j,t}^c, p_t, q_t) + e_{3,t} \\
&\approx h^c\left(\hat{m}_{j,t-1}^c, p_t, q_t\right) - H_{j,t}\hat{m}_{j,t-1}^c + e_{3,t}, \quad j = 1, \ldots, M_t,
\end{align*}
\]

where the Jacobian matrix \( H_{j,t} \) is straightforwardly computed using the chain rule, i.e.,
\[
H_{j,t} = \frac{\partial h^c}{\partial m_j^c} = \frac{\partial h^c}{\partial m_j^c} \frac{\partial m_j^c}{\partial m_j}.
\]
true trajectory is depicted. Note that the motion is only in the horizontal plane, i.e., with a fixed height, and with constant orientation.

C. Experimental Results

In this section results from the experiment are presented. In Figure 3 the result from dead-reckoning, i.e., direct double integration of the IMU acceleration data (after appropriate rotation from sensor to the earth-fixed system) is depicted together with the estimated trajectory using the MPF-SLAM method. The ground truth is provided from accurate measurements in the robot. Since the motion was rather smooth, high frequency dynamics in the robot arm was not excited, hence yielding a very accurate ground truth value. As can be seen the performance in position is significantly improved when the MPF-SLAM method is used, compared to only dead-reckoning. The reason is the filter, where the orientation and acceleration coupling together with the map features improve the estimates. In the current experiment the bias terms were not used.

In Figure 4 the measurement from the accelerometers, the true acceleration (from the industrial robot), the measured acceleration (from the IMU sensor) and the estimates from Algorithm 1. MPF-SLAM, and the ground truth from the industrial robot are depicted for the experiment. The vision system is used to build up the map (feature tracks of landmarks), which is depicted in Figure 5 for a specific time. Note that the uncertainty is visible in the spread of the particle cloud.

### Table I

<table>
<thead>
<tr>
<th>System, Filter, and Sensor Parameters for the Experiment.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Noise</strong></td>
</tr>
<tr>
<td>Cov. Acceleration $Q_a = \text{diag}(0.5^2, 0.5^2, 0.5^2)$</td>
</tr>
<tr>
<td>Cov. Bias angular rate $Q_{b_w} = \text{diag}(10^{-10}, 10^{-10}, 10^{-10})$</td>
</tr>
<tr>
<td>Cov. Bias acceleration $Q_{b_a} = \text{diag}(0.001^2, 0.001^2, 0.001^2)$</td>
</tr>
<tr>
<td>Cov. Angular rate $Q_w = \text{diag}(0.001^2, 0.001^2, 0.001^2)$</td>
</tr>
<tr>
<td><strong>Measurement Noise</strong></td>
</tr>
<tr>
<td>Cov. Camera $R_c = \text{diag}(0.01^2, 0.01^2)$</td>
</tr>
<tr>
<td>Cov. Accelerometer $R_a = \text{diag}(0.02^2, 0.02^2, 0.03^2)$</td>
</tr>
<tr>
<td>Cov. Gyroscope $R_w = \text{diag}(0.02^2, 0.03^2, 0.03^2)$</td>
</tr>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td>Sample freq. (IMU) $1/T = 100$ Hz</td>
</tr>
<tr>
<td>Sample freq. camera $1/T_c = 12.5$ Hz</td>
</tr>
<tr>
<td>Camera resolution $340 \times 280$ pixels</td>
</tr>
<tr>
<td>Number of particles $N = 100$</td>
</tr>
</tbody>
</table>

Figure 3. The ground truth position from the industrial robot, dead-reckoned position from inertial sensor data (5 seconds in 100 Hz), the estimate from the MPF-SLAM algorithm with vision measurements.

Figure 4. The true acceleration (from the industrial robot), the measured acceleration (from the IMU sensor) and the estimates from Algorithm 1.

Figure 5. Vision measurements from the feature extraction (cross) and current map represented by the particle cloud (red dots). Please note that this figure has to be viewed in color.

V. Conclusion

This contribution has introduced a Rao-Blackwellized/marginalized particle filtering framework for solving the SLAM problem, enabling high dimensional state vectors for the moving platform. The idea draws on the FastSLAM algorithm, but rather than using only the map, we also include all the states corresponding to a linear Gaussian sub-structure in the analytically solvable Kalman filter part. The approach was validated using a simple application.
example, where inertial measurements were fused with vision measurements. The correctness of the result was assessed using the ground truth.

**ACKNOWLEDGMENT**

The authors would like to thank MOVIII (Modeling, Visualization and Information Integration), the Swedish research council (VR) and the MATRIS project (EU research project, contract number IST-002013) for funding this work.

**APPENDIX A COORDINATE SYSTEMS**

The following convention is used to rotate a vector from a coordinate system \( A \) to a coordinate system \( B \),

\[ x_B = R_{BA} x_A, \]

where \( R_{BA} \) is used to denote the rotation matrix describing the rotation from \( A \) to \( B \). Hence, we can get from system \( A \) to \( C \), via \( B \) according to

\[ R_{CA} = R_{CB} R_{BA}. \]

This can also be expressed using unit quaternions \( q \),

\[ u_b = \bar{q}_a \odot u_a \odot q, \]

where \( u_a \) is the quaternion extension of the vector \( x_A \), i.e., \( u_a = (0, x^T_A) \) and \( \odot \) represents quaternion multiplication. Furthermore, \( \bar{u} \) denotes the quaternion conjugate. See e.g., [32], [33] for an introduction to unit quaternions and other rotation parameterizations.

It is straightforward to convert a given quaternion into the corresponding rotation matrix,

\[ R(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_1 q_0) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_1 q_0) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}. \]

The following coordinate systems are used in this paper. An earth-fixed (denoted with \( e \)), body or inertial sensor system (denoted with \( b \)), and camera system (denoted with \( c \)).

**REFERENCES**


