A Particle Filter for Monocular Vision-Aided Odometry

Teddy Yap, Jr
Washington State University, Pullman
Email: tyap@eecs.wsu.edu

Mingyang Li and Anastasios I. Mourikis and Christian R. Shelton
University of California, Riverside
Email: mli@ee.ucr.edu, mourikis@ee.ucr.edu, cshelton@cs.ucr.edu

***Abstract*— We propose a particle filter-based algorithm for monocular vision-aided odometry for mobile robot localization. The algorithm fuses information from odometry with observations of naturally occurring static point features in the environment. A key contribution of this work is a novel approach for computing the particle weights, which does not require including the feature positions in the state vector. As a result, the computational and sample complexities of the algorithm remain low even in feature-dense environments. We validate the effectiveness of the approach extensively with both simulations as well as real-world data, and compare its performance against that of the extended Kalman filter (EKF) and FastSLAM. Results from the simulation tests show that the particle filter approach is better than these competing approaches in terms of the RMS error. Moreover, the experiments demonstrate that the approach is capable of achieving good localization accuracy in complex environments.**

I. Introduction

For mobile robots to effectively navigate their surroundings, they must have precise estimates of their position and orientation (pose). To obtain such estimates, in most cases a robot fuses measurements from multiple onboard sensors. Typically, a *proprioceptive* sensor (e.g. odometry) provides estimates for the robot's motion, and one or more *exteroceptive* sensors (e.g. laser rangefinders) are used to provide additional localization information and improve the accuracy of the estimates. Recently, the use of *visual sensing* for navigation has been at the focus of extensive research efforts. Cameras are an attractive option for a number of reasons: They are cheap, small, low powered, and lightweight; they are becoming increasingly common; and they are versatile in that they can also be used for other robotic tasks (e.g. object or place detection and recognition). For navigation applications, cameras can be used to detect and track features in the environment, and the localization information in these observations can be fused with the input from proprioceptive sensors for ego-motion estimation.

In this paper, we propose a particle filter (PF)-based algorithm for monocular vision-aided odometry. The algorithm fuses information from odometry with observations of naturally occurring point features in the environment, taken by a camera mounted on the robot. Our key contribution is a novel approach for computing the particle weights, which does not require including the feature positions in the state vector. Unlike simultaneous localization and mapping (SLAM) techniques, our approach explicitly marginalizes out the features. This helps keep the size of the state vector small, resulting in low computational cost and a small number of particles required to represent the posterior distribution.

Existing methods that can be employed for vision-aided odometry utilize the assumption that the error in the camera pose estimates, or the error in the feature estimates, or both, is well-approximated by a Gaussian pdf. Specifically, EKF-SLAM methods (e.g., [1]) assume a Gaussian pdf for both the feature and the camera pose errors, sliding-window methods (e.g. [2]–[4]) do so for the errors of the marginalized camera poses, and FastSLAM [5] assumes that each feature's position pdf is approximately Gaussian, given the camera trajectory. Moreover, all these methods rely on the linearization of the camera measurement model, and thus require that the nonlinearity of this model is not significant.

In this work we remove these assumptions, and present an algorithm that makes no assumption on the nature of the pdfs arising in the estimation process. This algorithm employs the PF for estimation, and at each time step uses the measurements of all the features currently tracked for updating the particle weights. A key property of the proposed method is that its computational cost is linear in the number of locally visible features, which is crucial for achieving operation in feature-dense environments with limited computational resources. Our simulation results and real-world experiments demonstrate the effectiveness of the proposed approach, and show that it outperforms competing methods for the problem of real-time motion estimation.

II. Related Work

We here discuss related approaches employing monocular cameras for aiding navigation. Due to the limited space we only discuss some representative approaches, grouped into three broad categories: those that use a known map of the environment, SLAM approaches, and methods that do not require a map of the environment and do not perform mapping (such as our own).

a) Known-map based methods: One family of approaches for vision-aided navigation assumes the existence of an *a priori* feature map of the area where the robot operates. Examples of such methods include [6]–[8] and references therein. Observing known features provides absolute position information, and thus methods based on this paradigm are capable of drift-free pose estimation. However, the requirement for having a feature map limits the applicability of these algorithms to previously explored environments. In this work, we instead focus on the more challenging case where the feature locations are *not* known in advance, which permits operation in new and unknown environments.

b) SLAM methods: In order to avoid the need for an *a priori* map of the environment, one group of methods follows the SLAM framework. In this paradigm the robot's pose and the landmarks' positions are estimated jointly, typically in a recursive-filtering framework (e.g [1], [9]–[11] and references therein). While generating a feature map may be useful for certain tasks, when such a map is not necessary, allocating computing and memory resources to the updating of an ever-increasing number of features is not desirable. Thus, in this work we focus on the problem of estimating the robot trajectory *without* maintaining a feature map.

c) Visual-odometry methods: Methods that only estimate the camera pose and forgo including the feature positions in the state vector are often collectively termed *visual odometry* algorithms. In these approaches, visual feature observations are utilized to derive probabilistic constraints (i.e., measurements) between the camera poses from which the features where observed. In their most computationally efficient (and simplest) form, visual odometry methods use the feature observations to derive relative-pose measurements between two consecutive time instants [12]–[14]. This approach results in real-time performance, but is suboptimal in the (common) case where each feature is seen in more than two images.

This problem can be addressed by employing the feature measurements to extract relative-pose information for the entire *sliding window* of poses from which the feature is observed. In this way, all the motion information contained in the feature observations can be used. Existing methods that follow this paradigm are typically based either on the EKF estimator [2], or on an iterative least-squares methodology [3], [4], [15]. However, all these methods require the linearization of the measurement models (and thus, assume only moderate non-linearities in these models), and all employ the assumption that the pdf of the marginalized robot poses is Gaussian. When these assumptions are not valid (e.g., when estimation errors grow large), the use of linearization-based approaches is not suitable.

For these reasons, in this work we propose a PF-based estimator for visual odometry. Owing to the PF's inherent ability to represent non-Gaussian distributions, this approach is better suited for scenarios where the estimation errors are significant, and/or the measurement models are highly nonlinear. We note that one straightforward approach for employing the particle filter to the problem at hand would be to use the FastSLAM algorithm [5], and to drop all features from the state vector once they disappear from the field of view. However, FastSLAM still employs an EKF for each landmark, and thus its performance depends on the quality of linearization. Contrary to that, our algorithm explicitly marginalizes out the features, and the particle weight updates do not rely on precise linearization. In the results presented in Section IV we show that the proposed method outperforms FastSLAM, especially when features can only be observed for a small number of frames.

III. Particle Filtering Approach

A. Overview

The objective of our PF-based estimator is to track the pose of a wheeled robot using odometry and the observations of static point features through its camera. We achieve this goal by using a PF to track the sequence of the last N_p robot poses. Conditioned on these poses, each feature's observations are independent of the observations of other features. However, we do not keep an explicit posterior distribution over feature positions conditioned on each particle, as is done in FastSLAM. Instead, we marginalize out each feature's position to obtain the probability of the sequence of that feature's observations, conditioned on the particle's sequence of poses. This is used to update the weight of the particle, as normal for a PF.

The mechanism for performing the explicit marginalization of the features is the first important contribution of this paper. Specifically, the marginalization amounts to the evaluation of an expectation with respect to the feature position (see Eq. 3), which cannot be computed in closed form. To address this problem, we employ the unscented transform [16]. As the unscented transform is designed to estimate an expectation with respect to a Gaussian distribution (and the true posterior is not normally distributed), we introduce a "proposal" Gaussian distribution over the position of the feature. This proposal distribution is estimated from the feature measurements, making it similar to the posterior feature position distribution. However, it is *not* assumed to be the true distribution and is only used as a catalyst for computing the measurement likelihood.

The second important contribution of this paper is in the incremental update of the weights of the particles. Normally, one would use each feature's measurements only once to update the particle weights, after the feature is lost from view. However this is not desirable, since it results in delayed use of information. In our framework, we update the weight of the particle based on a feature at *every* time step. If $\lambda_t^{(j)[i]}$ is the weight contribution of feature j to particle i at time t (i.e. the probability of the entire sequence of observations of feature j given particle i), then we multiply the weight of particle i by $\lambda_t^{(j)[i]}/\lambda_{t-1}^{(j)[i]}$ to cancel out previous weight updates and prevent over-counting of earlier evidence. At first glance, this would appear to accomplish nothing: the final weight change for a particle due to a feature is the same. However, if the particles are resampled in the middle of the track of the feature, this can make a large difference. Using the evidence as it becomes available, instead of only after a feature track is complete, results in a more "focused" posterior distribution, and improves the performance of resampling.

Taken together, these two non-standard modifications allow us to effectively use a PF to model the non-linearities in the motion and observation models.

B. Notation

We assume that the robot operates on a planar environment so that the robot pose at time t can be represented by a vector $\mathbf{x}_t = [x_t, y_t, \theta_t]^T$, where $[x_t, y_t]^T$ are its two-dimensional Cartesian coordinates and θ_t is its orientation with respect to some global reference frame G (which we choose to coincide with the initial robot frame).

Each particle in our filter maintains an estimate of the recent N_p robot poses at time t. Let the ith particle at time t be $\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t} \triangleq \left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}+1}, \mathbf{x}^{[i]}_{t-N_\mathrm{p}+2}, ..., \mathbf{x}^{[i]}_{t}\right)$. Maintaining this window of poses in the particle estimates is necessary, as all measurements of each feature are used collectively in the marginalization process. Feature measurements are incrementally processed for features that are tracked for at least two and up to a maximum of N_p consecutive frames. A feature that is tracked for more than N_p frames will be considered as a new feature starting at frame number $(N_\mathrm{p}+1)$. Let $\mathcal{X}_t = \left\{\left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}, w^{[i]}_t\right)\right\}_{i=1}^{N_\mathrm{s}}$ be the set of weighted particles at time t, where $w^{[i]}_t$ are the particle weights, and N_s is the number of particles in the filter.

For feature observations, let f_j be a feature, and C_k be the kth camera frame. The position vector of the feature with respect to C_k is denoted by $\left[{}^{C_k}x_{f_j}, {}^{C_k}y_{f_j}, {}^{C_k}z_{f_j}\right]^T = {}^{C_k}\mathbf{p}_{f_j}$. As explained in Section III-E, in this work we employ an inverse-depth parameterization of the feature position, so ${}^{C_k}\mathbf{p}_{f_j}$ is parameterized by a 3-vector $\mathbf{f}_j$. We represent camera frame C_k as $\{{}^G_{C_k}\mathbf{R}, {}^G\mathbf{p}_{C_k}\}$, where ${}^A_B\mathbf{R}$ is the 3×3 matrix describing the rotation between frames A and B, and ${}^A\mathbf{p}_B$ is the 3D position of the origin of frame B with respect to frame A. We are assuming a calibrated camera, and thus the observation of feature f_j in camera frame C_k is described by the pinhole camera model:

$$\mathbf{o}^{(j)}_k = \frac{1}{{}^{C_k}z_{f_j}} \begin{bmatrix} {}^{C_k}x_{f_j} \\ {}^{C_k}y_{f_j} \end{bmatrix} + \mathbf{n}^{(j)}_k \ , \tag{1}$$

where $\mathbf{n}^{(j)}_k$ is the 2×1 image noise vector, modelled as a random variable with known distribution.

Let $\mathcal{S}_{f_j,t}$ be the set of camera frames in which feature f_j is observed, and $M_{f_j,t}$ be the size of this set. Then the set of all the measurements of feature f_j observed through time t is denoted by $\mathcal{O}^{(j)}_t \triangleq \left\{\mathbf{o}^{(j)}_k : k \in \mathcal{S}_{f_j,t}\right\}$.

C. Motion Model

We do not make any assumptions about the underlying motion model for the robot, other than that we can sample from it. The particles are propagated by sampling their current pose given their previous pose and the control measurements $\mathbf{c}_t$, i.e. $\mathbf{x}^{[i]}_t \sim p\left(\mathbf{x}_t \,\middle|\, \mathbf{x}^{[i]}_{t-1}, \mathbf{c}_t\right)$, utilizing the known statistics of the errors.

D. Weight Updates

In the particle filtering framework, the weight of each particle is proportional to the likelihood of the observations given the particle estimates. In our approach, all the measurements of a feature through time t are used simultaneously to update the particle weights. Specifically, processing the measurements of feature f_j will result in updating the weight of particle i by a factor equal to

$$\lambda^{(j)[i]}_t = p\left(\mathcal{O}^{(j)}_t \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right) \ . \tag{2}$$

Note that the above cannot be directly computed, since the observation is not conditioned on the position of the feature f_j. We introduce the feature position as follows:

$$\lambda^{(j)[i]}_t = \int p\left(\mathcal{O}^{(j)}_t \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}, \mathbf{f}_j\right) p\left(\mathbf{f}_j \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right) d\mathbf{f}_j \ . \tag{3}$$

The first term in the above integrand is the measurement likelihood function, which depends on the projection geometry and the noise model. For a given camera and experimental setup, this likelihood function can be computed analytically (more details in Sections III-E and III-F).

We assume that, when no measurement information is available, the feature can be anywhere in the 3D space with equal probability (i.e., an "uninformative" prior). We can thus write $p\left(\mathbf{f}_j \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right) = \gamma\left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right)$, which leads to:

$$\lambda^{(j)[i]}_t = \gamma\left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right) \int p\left(\mathcal{O}^{(j)}_t \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}, \mathbf{f}_j\right) d\mathbf{f}_j \ . \tag{4}$$

The derivation of the term $\gamma\left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right)$ is presented in detail in Section III-H.

In order to compute the integral in Eq. 4, we introduce a proposal distribution $q(\mathbf{f}_j)$ over the potential feature position $\mathbf{f}_j$, which leads to

$$\begin{aligned}\lambda^{(j)[i]}_t &= \gamma\left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right) \int \frac{p\left(\mathcal{O}^{(j)}_t \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}, \mathbf{f}_j\right)}{q(\mathbf{f}_j)} q(\mathbf{f}_j)\, d\mathbf{f}_j \\ &= \gamma\left(\mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}\right) \cdot \mathbb{E}_{q(\mathbf{f}_j)}\left[\frac{p\left(\mathcal{O}^{(j)}_t \,\middle|\, \mathbf{x}^{[i]}_{t-N_\mathrm{p}:t}, \mathbf{f}_j\right)}{q(\mathbf{f}_j)}\right]\end{aligned} \tag{5}$$

While the proposal $q(\mathbf{f}_j)$ can be any distribution, it is convenient to choose $q(\mathbf{f}_j)$ to be the Gaussian distribution $\mathcal{N}\left(\hat{\mathbf{f}}_j, \mathbf{C}^{(j)}\right)$, where $\hat{\mathbf{f}}_j$ is an estimated position of feature f_j and $\mathbf{C}^{(j)}$ is the associated covariance matrix of the estimate. Now the integral in Eq. 5 amounts to an expectation with respect to a Gaussian random variable, which can be evaluated by application of the unscented transform [16], [17]. We defer the discussion of how to select $\hat{\mathbf{f}}_j$ and $\mathbf{C}^{(j)}$ to Section III-G.

Eq. 5 provides the contribution of feature f_j to the weight of the ith particle at time t. Considering all the features that have been observed at time t, the weight $w^{[i]}_t$ of the ith particle can be updated as

$$w^{[i]}_t = w^{[i]}_{t-1} \prod_j \frac{\lambda^{(j)[i]}_t}{\lambda^{(j)[i]}_{t-1}} \ . \tag{6}$$

Note that in Eq. 6, we divided $\lambda^{(j)[i]}_t$ by $\lambda^{(j)[i]}_{t-1}$, which is the contribution of feature f_j to the weight of the ith particle at time $t-1$. This is done to avoid "double counting" the contribution of feature f_j to the weight of the ith particle, in case the observations of feature f_j were processed at the previous time step $t-1$ (if the observations of feature f_j are processed for the first time at time t, then we set $\lambda^{(j)[i]}_{t-1} = 1$). With this weighting scheme, feature observations are processed incrementally at every time step without waiting for the features to be lost before their measurements are processed. Note that while $w^{[i]}_{t-1}$ may have been changed (due to the resampling step of the particle filter), the past weight contributions for the features, $\lambda_{t-1}^{(j)[i]}$, were not. Therefore, Eq. 6 does not directly reduce to $\prod_j \lambda_t^{(j)[i]}$.

E. Feature Parameterization

The parameterization of feature f_j is critical to the success of using a Gaussian proposal distribution $q(\mathbf{f}_j)$. In previous work [2], [9], it has been demonstrated that an inverse-depth parameterization has a key advantage over the traditional Cartesian-coordinate parameterization: the measurement model exhibits better linearity in the inverse-depth parameters, and thus the resulting estimates are better approximated by a Gaussian. Therefore, in this work we employ an inverse-depth parameterization, as discussed next.

The position of feature f_j in camera frame C_k is given by

$$^{C_k}\mathbf{p}_{f_j} = \begin{bmatrix} ^{C_k}x_{f_j} \\ ^{C_k}y_{f_j} \\ ^{C_k}z_{f_j} \end{bmatrix} = {}^C_R\mathbf{R}\,{}^G_{R_k}\mathbf{R}^T({}^G\mathbf{p}_{f_j} - {}^G\mathbf{p}_{R_k}) + {}^C\mathbf{p}_R \ ,$$

where

$${}^G_{R_k}\mathbf{R} = \begin{bmatrix} \cos(\theta_k) & -\sin(\theta_k) & 0 \\ \sin(\theta_k) & \cos(\theta_k) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \ {}^G\mathbf{p}_{R_k} = \begin{bmatrix} x_k \\ y_k \\ h \end{bmatrix}, \quad (7)$$

h is the known height of the robot, ${}^G\mathbf{p}_{f_j}$ is the three-dimensional position of feature f_j in the global frame G, and $\{{}^C_R\mathbf{R}, {}^C\mathbf{p}_R\}$ is the known transform between the camera frame C and the robot frame R. Since the three-dimensional position ${}^G\mathbf{p}_{f_j}$ of the feature in the global frame G is unknown, we need to first estimate it using the measurements $\mathbf{o}_k^{(j)}$, $k \in \mathcal{S}_{f_j,t}$, and the $M_{f_j,t}$ camera poses where the feature was observed. However, instead of directly computing an estimate of the three-dimensional position of the feature in the global frame, we compute an estimate of the inverse-depth parameterization of the feature with respect to C_n, the last camera frame in which the feature was observed. The feature coordinates with respect to the kth camera frame are

$$\begin{aligned} {}^{C_k}\mathbf{p}_{f_j} &= {}^{C_k}_{C_n}\mathbf{R}\,{}^{C_n}\mathbf{p}_{f_j} + {}^{C_k}\mathbf{p}_{C_n} && (8) \\ &= {}^{C_n}z_{f_j}\left({}^{C_k}_{C_n}\mathbf{R}\begin{bmatrix} \frac{^{C_n}x_{f_j}}{^{C_n}z_{f_j}} \\ \frac{^{C_n}y_{f_j}}{^{C_n}z_{f_j}} \\ 1 \end{bmatrix} + \frac{1}{^{C_n}z_{f_j}}{}^{C_k}\mathbf{p}_{C_n}\right) && (9) \\ &= {}^{C_n}z_{f_j}\left({}^{C_k}_{C_n}\mathbf{R}\begin{bmatrix} \alpha_j \\ \beta_j \\ 1 \end{bmatrix} + \rho_j\,{}^{C_k}\mathbf{p}_{C_n}\right) && (10) \\ &= {}^{C_n}z_{f_j}\begin{bmatrix} \psi_{k1}(\alpha_j,\beta_j,\rho_j) \\ \psi_{k2}(\alpha_j,\beta_j,\rho_j) \\ \psi_{k3}(\alpha_j,\beta_j,\rho_j) \end{bmatrix} \ , && (11) \end{aligned}$$

where α_j, β_j, and ρ_j are the inverse-depth parameters of feature f_j with respect to the camera frame C_n, defined as

$$\alpha_j \triangleq \frac{^{C_n}x_{f_j}}{^{C_n}z_{f_j}} \ , \quad \beta_j \triangleq \frac{^{C_n}y_{f_j}}{^{C_n}z_{f_j}} \ , \quad \rho_j \triangleq \frac{1}{^{C_n}z_{f_j}} \ , \qquad (12)$$

and ψ_{k1}, ψ_{k2}, and ψ_{k3} are functions of α_j, β_j, and ρ_j. The vector $\mathbf{f}_j = [\alpha_j, \beta_j, \rho_j]^T$ defines the parameterization of the feature. We can now express the observation $\mathbf{o}_k^{(j)}$ at time step k as:

$$\begin{aligned} \mathbf{o}_k^{(j)} &= \frac{1}{\psi_{k3}(\alpha_j,\beta_j,\rho_j)}\begin{bmatrix} \psi_{k1}(\alpha_j,\beta_j,\rho_j) \\ \psi_{k2}(\alpha_j,\beta_j,\rho_j) \end{bmatrix} + \mathbf{n}_k^{(j)} \\ &= \psi_k(\mathbf{f}_j) + \mathbf{n}_k^{(j)} \ . \end{aligned} \qquad (13)$$

F. Noise model

We now present the form of the likelihood function appearing in Eqs. 3-5. This function accounts for the presence of *outliers*, which is an important concern that needs to be addressed in any vision system. Outliers can be caused by several factors (e.g., moving objects, tracking failures), and if care is not taken, even a small number can cause estimator failure. To model the existence of outliers, we employ the total probability theorem to obtain

$$\begin{aligned} p\left(\mathcal{O}_t^{(j)}\middle|\mathbf{x}_{t-N_\mathrm{p}:t},\mathbf{f}_j\right) &= p(\mathrm{In})p\left(\mathcal{O}_t^{(j)}\middle|\mathbf{x}_{t-N_\mathrm{p}:t},\mathbf{f}_j,\mathrm{In}\right) \\ &\quad + p(\mathrm{Out})p\left(\mathcal{O}_t^{(j)}\middle|\mathbf{x}_{t-N_\mathrm{p}:t},\mathbf{f}_j,\mathrm{Out}\right) \end{aligned} \qquad (14)$$

where $p(\mathrm{In})$ is the probability that a feature is an inlier (i.e., a correctly tracked static feature), and $p(\mathrm{Out}) = 1 - p(\mathrm{In})$ is the probability that a feature is an outlier. For inlier features we assume that the measurement noise in each image is independent, zero mean, and Gaussian, with standard deviation σ_im, and therefore

$$p\left(\mathcal{O}_t^{(j)}\middle|\mathbf{x}_{t-N_\mathrm{p}:t},\mathbf{f}_j,\mathrm{In}\right) = \prod_{k\in\mathcal{S}_{f_j,t}} \mathcal{N}\left(\mathbf{o}_k^{(j)} - \psi_k(\mathbf{f}_j);\mathbf{0},\sigma_\mathrm{im}^2\mathbf{I}_2\right)$$

where $\mathcal{N}\left(\cdot;\mathbf{0},\sigma_\mathrm{im}^2\mathbf{I}_2\right)$ denotes the Gaussian pdf with zero mean and covariance $\sigma_\mathrm{im}^2\mathbf{I}_2$ ($\mathbf{I}_2$ is the 2×2 identity matrix).

The pdf for outlier features may depend on the feature tracker implementation and the properties of the environment. For simplicity, we model the image noise in the case of outliers by a Gaussian pdf with large standard deviation, $\sigma_\mathrm{im}^{out} \gg \sigma_\mathrm{im}$. In addition to the outlier noise model, the likelihood function in Eq. 14 requires knowledge of the percentage of outliers in the data (i.e., of $p(\mathrm{In})$ and $p(\mathrm{Out})$). This can either be learned from data, or estimated based on the properties of the implementation and the environment.

G. Feature Estimation Using Gauss-Newton Minimization

In order to estimate the inverse-depth parameterization of feature f_j we use a maximum-likelihood estimator (MLE). In this estimator the robot pose estimates $\mathbf{x}_{t-N_\mathrm{p}:t}^{[i]}$ are treated as known constants, and the unknown parameter to be estimated is $\mathbf{f}_j$. For this step, each feature is treated as an inlier, to simplify the implementation and also because the position of outlier features is not always well-defined. This leads to a Gaussian likelihood function, and allows us to re-formulate the MLE into a nonlinear least squares problem, which is solved by employing Gauss-Newton iterative minimization [18]. The output of this step is the estimated inverse-depth parameterization $\hat{\mathbf{f}}_j$ and its associated covariance matrix $\mathbf{C}^{(j)}$. In our implementation $\hat{\mathbf{f}}_j$ is computed for each particle in the filter, since each particle represents a different estimate of the camera trajectory.

We reiterate that this Gaussian estimate of the position of a feature is not directly used by the PF as an estimate of the feature's position. Rather, it is used as a proposal distribution for employing the unscented transform to marginalize out the position of the feature. Thus, even if due to inaccurate linearization the estimate of $\mathbf{C}^{(j)}$ is imprecise, this will not adversely affect the performance of the filter. We have found that the resulting filter is quite robust to different methods for selecting $\hat{\mathbf{f}}_j$ and $\mathbf{C}^{(j)}$.

H. Derivation of the prior term

We now present the derivation of the term $\gamma\left(\mathbf{x}^{[i]}_{t-N_\text{p}:t}\right)$, which expresses the prior on the feature's position (see Eq. 4). This derivation is based on the key observation that, if two particles differ only in scale, then the weights computed for these two particles should be identical. This is dictated by the fact that the camera measurements alone cannot provide scale information. Consider two different particles, $\mathbf{x}^{[i]}_{t-N_\text{p}:t}$ and $\mathbf{x}^{[i']}_{t-N_\text{p}:t}$, that differ only in the scale of the trajectory (i.e., $x^{[i]}_k = sx^{[i']}_k$, $y^{[i]}_k = sy^{[i']}_k$ and $\theta^{[i]}_k = \theta^{[i']}_k$ for $k = t-N_\text{p}+1, \ldots t$, where s is a constant scaling factor). For a given feature f_j, and for the particle $\mathbf{x}^{[i']}_{t-N_\text{p}:t}$, we have

$$\lambda^{(j)[i']}_t = \gamma\left(\mathbf{x}^{[i']}_{t-N_\text{p}:t}\right) \int p\left(\mathcal{O}^{(j)}_t \left| \mathbf{x}^{[i']}_{t-N_\text{p}:t}, \mathbf{f}_j\right.\right) d\mathbf{f}_j \qquad (15)$$

Recall that $\mathbf{f}_j = [\alpha_j, \beta_j, \rho_j]^T$. Thus, by employing a change of variable in the above integral, to use $\mathbf{f}'_j = [\alpha_j, \beta_j, s\rho_j]^T$, and noting that $p\left(\mathcal{O}^{(j)}_t \left| \mathbf{x}^{[i']}_{t-N_\text{p}:t}, \mathbf{f}'_j\right.\right) = p\left(\mathcal{O}^{(j)}_t \left| \mathbf{x}^{[i]}_{t-N_\text{p}:t}, \mathbf{f}_j\right.\right)$, we obtain:

$$\begin{aligned}\lambda^{(j)[i']}_t &= s\,\gamma\left(\mathbf{x}^{[i']}_{t-N_\text{p}:t}\right) \int p\left(\mathcal{O}^{(j)}_t \left| \mathbf{x}^{[i]}_{t-N_\text{p}:t}, \mathbf{f}_j\right.\right) d\mathbf{f}_j \\ &= s\frac{\gamma\left(\mathbf{x}^{[i']}_{t-N_\text{p}:t}\right)}{\gamma\left(\mathbf{x}^{[i]}_{t-N_\text{p}:t}\right)} \lambda^{(j)[i]}_t \end{aligned} \qquad (16)$$

From the above expression we see that, to ensure $\lambda^{(j)[i']}_t = \lambda^{(j)[i]}_t$ (as needed for scale invariance), the term $\gamma(\cdot)$ must be proportional to the scale of the trajectory. To achieve this, we choose $\gamma\left(\mathbf{x}^{[i]}_{t-N_\text{p}:t}\right)$ to be equal to the maximum distance between any two camera poses in the sliding window represented by the ith particle (i.e., equal to the baseline).

I. Summary of Particle Weighting

The following summarizes the steps needed to assign weights to the particles.

- For each particle $\mathbf{x}^{[i]}_{t-N_\text{p}:t}, i = 1, 2, ..., N_\text{s}$
  - For each feature f_j tracked at time t
    - ∗ Use $\mathcal{O}^{(j)}_t$, $\mathbf{x}^{[i]}_{t-N_\text{p}:t}$, and the Gauss-Newton minimization to obtain an estimate $\hat{\mathbf{f}}_j$ and its associated covariance $\mathbf{C}^{(j)}$.
    - ∗ Compute $\lambda^{(j)[i]}_t$, the contribution of feature f_j to the weight of the ith particle, using Eq. 5 with $q(\mathbf{f}_j) = \mathcal{N}\left(\hat{\mathbf{f}}_j, \mathbf{C}^{(j)}\right)$.
  - Compute the overall weight $w^{[i]}_t$ using Eq. 6.

J. Selective Resampling

In order to reduce the risk of particle depletion in the PF, we resample from the set of weighted particles only if the number of effective samples

$$\hat{N}_\text{eff} = \left(\sum_{i=1}^{N_s} w^{[i]}_t\right)^2 \Bigg/ \sum_{i=1}^{N_s} \left(w^{[i]}_t\right)^2 \qquad (17)$$

falls below a certain threshold N_τ [19].

IV. Experimental Results

A. Simulation Tests

The goal of the simulation experiments is to test the performance of the particle filtering approach under different conditions by varying certain parameters. Fig. 1 shows our simulation environment, which is a 12m $\times$ 12m $\times$ 5m room with two hundred visual point features randomly placed on the walls. The robot is moving along a circular path of radius 3m with a constant velocity and angular velocity of 0.1m/s and 0.0333rad/s, respectively. The measured velocity and angular velocity are corrupted by independent zero-mean Gaussian noise with standard deviations 0.01m/s and $1°$/s, respectively. As the robot moves, its camera records images at 1Hz. We assume that the camera has a field of view of $47.5°$ and $\sigma_\text{im} = 1/400$ (this corresponds to standard deviation of one pixel, in a camera with a focal length equal to 400 pixels). The duration of the simulation was set to 1000 seconds.

The proposed PF-based algorithm is compared against the MSCKF algorithm [2] and FastSLAM [5]. The MSCKF employs a similar technique of explicit feature marginalization, but uses an EKF estimator for this purpose. On the other hand, FastSLAM makes use of a Rao-Blackwellized particle filter, with an EKF for each feature. In our tests, all three filters process the same feature tracks, and use the same inverse-depth parameterization. Moreover, for fairness of comparison, in our implementation of FastSLAM features are only tracked for a maximum of N_p states, and if a feature is seen again after a full circle, it is treated as a new one – the same as in our PF and in the MSCKF. To evaluate the filters' performance, we compute the root mean squared error (RMSE) over the 100 Monte Carlo trials for all three algorithms. In our current implementation, FastSLAM is four times faster than our PF, for the same number of particles. Therefore, to ensure a fair comparison in which the two methods are allocated the same amount of CPU time, we use four times fewer particles in our method, compared to FastSLAM.

We experimented with different values for N_p and different number of particles N_s for the particle-filter based methods, and the results are presented in Table I. These results show that the PF performs better than the MSCKF for all the cases

TABLE I

RMSE OF THE MSCKF, OUR PF, AND FASTSLAM, OVER 100 MONTE CARLO TRIALS.

	MSCKF	Particle Filter			FastSLAM			
		N_s			N_s			
N_p		250	500	1000	1000	2000	4000	
2	0.679	0.406	0.361	0.370	0.470	0.466	0.493	x
	0.656	0.376	0.342	0.330	0.514	0.522	0.511	y
	0.208	0.110	0.098	0.094	0.123	0.115	0.126	θ
3	0.462	0.305	0.303	0.281	0.331	0.338	0.363	x
	0.468	0.292	0.286	0.248	0.350	0.369	0.345	y
	0.154	0.062	0.067	0.060	0.067	0.066	0.069	θ
5	0.315	0.254	0.246	0.245	0.289	0.290	0.274	x
	0.317	0.256	0.244	0.258	0.282	0.264	0.274	y
	0.096	0.038	0.039	0.041	0.042	0.040	0.037	θ
10	0.245	0.230	0.232	0.216	0.226	0.238	0.228	x
	0.229	0.208	0.237	0.226	0.233	0.230	0.239	y
	0.071	0.023	0.027	0.018	0.025	0.022	0.023	θ

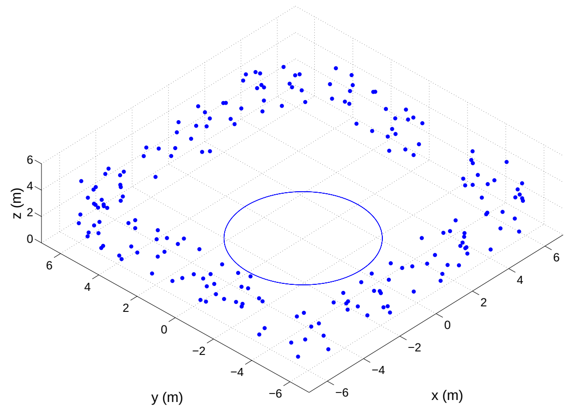

Fig. 1. The simulation environment and the true trajectory of the robot (circle). Each dot represents a point landmark placed on the wall of the environment.

considered. As N_p increases, the performance of the EKF and PF are comparable, since the estimates are more accurate, and the linearization errors in the EKF are less severe. However, for small values of N_p the performance of the PF exhibits a more graceful degradation than that of the MSCKF. Similarly, the proposed PF algorithm is clearly better than FastSLAM for small values of N_p, while the two methods perform equally well for large N_p. To explain this behavior, we note that when features can only be tracked for a small number of states, the Gaussianity assumption employed by the EKFs in FastSLAM is a poor one. Consequently, the estimates produced by FastSLAM suffer from larger errors. As N_p increases, the feature estimate errors become "more Gaussian", and thus FastSLAM's performance improves.

As a final remark, we note that decreasing the number of particles in the PF does not significantly affect its performance. In the simulation scenario used here, decreasing the number of particles from 1000 to 250 results in an increase of the error by less than 15%. If using even fewer particles is desired, one could implement more advanced resampling schemes, e.g. the auxiliary particle filter [20].

B. *Real-World Experiment*

To demonstrate the effectiveness and applicability of the PF approach in a real-world setting, we apply the algorithm

Fig. 2. Sample images from the real-world experiment.

on a real image sequence collected with our mobile robot. In our experiment, we used an ActivMedia Robotics P3-DX equipped with wheel encoders for odometry and a front-looking Canon VC-C50i camera that provides images of resolution 320×240 pixels.

The experiment was conducted inside the Bourns Engineering Building II, and involved an approximately 200-m long trajectory. Some sample images are shown in Fig. 2. We process a new image whenever the robot's encoders indicate that the robot translated by at least 0.1m or rotated by at least 1°. In this dataset approximately 2200 images are processed. We pre-calibrated the camera using a Matlab camera calibration toolbox [21], and we used the Kanade-Lucas-Tomasi (KLT) tracker [22], [23] for extracting and tracking corner features. The percentage of outliers has been determined at 10% (i.e., $p(\text{Out}) = 0.1$), the standard deviation of the inlier feature measurements is 1 pixel, and $\sigma_{\text{im}}^{out} = 10\sigma_{\text{im}}$. For the results shown here, we used 1000 particles for our PF, and 4000 for FastSLAM.

Fig. 3 shows the trajectory estimates of the three filters considered (PF, MSCKF, FastSLAM), as well as the desired robot path. The robot was manually commanded along its path, starting at position $(0, 0)$. Similarly to the simulation results of the preceding section, in the real-world experiment the proposed PF algorithm outperforms both the MSCKF and FastSLAM in terms of accuracy. In addition, because both the PF and FastSLAM are not deterministic algorithms and their results vary by run, the standard deviation of the trajectory endpoint was computed over 100 trials. For the PF, the standard deviation is 0.8826m in the x axis and 1.3942m in the y axis, while for FastSLAM we obtain 1.5458m and 1.4366m, respectively. We thus see that in addition to being more accurate, the PF result is also more stable, with smaller standard deviation for its estimates. It should be noted that a large window size is chosen in this experiment ($N_p = 30$) to ensure that the motion information from features that are tracked over long periods is properly used. Overall, the attained estimation accuracy is satisfactory, given the fact that no loop closure or prior map information is used, and that the robot motion is mostly parallel to the camera's optical axis, which is a difficult scenario for vision-based motion

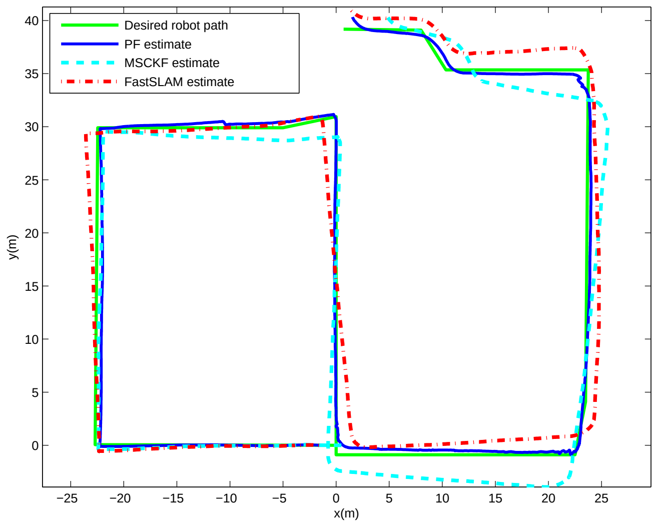

Fig. 3. The desired path for the robot (thin solid line – green), PF (thick solid line – blue), the MSCKF (thick broken line – magenta), and FastSLAM (dashdot line – red). The robot started at position $(0, 0)$.

estimation.

The processing of the experimental results presented above was carried out off-line. However, the proposed algorithm is capable of real-time operation. Our current implementation has been demonstrated to run at 7fps using 1000 particles on a low-end, 5-year old laptop (this includes both image processing for feature extraction and pose estimation). Using newer hardware and further code optimization will significantly increase the performance of the method.

V. Conclusions

Our proposed particle filter performs better than FastSLAM and the EKF for vision-aided odometry. We attribute this to the flexibility of the method to represent non-Gaussian distributions and non-linear functions. In particular we represent the distribution of the robot's position with a set of particles instead of as a Gaussian (like FastSLAM, but unlike the EKF). Additionally, we do not approximate the distribution over the features' position as a Gaussian (like the MSCKF [2], but unlike FastSLAM). Therefore, our method is more robust to non-linearities, which are common in visual odometry, especially when features are only tracked for a few frames.

We note that our method can be extended to other applications that combine ego-motion estimates with feature measurements, such as inertial measurements and images, or odometry and laser range data. We expect that modeling additional variables (such as an inertial sensor's biases and velocity) would require Rao-Blackwellized particle filters or other similar methods. However, the foundation of this paper in providing immediate weight updates (instead of waiting until a feature has been lost) and marginalizing out the feature positions still applies.

Acknowledgment

This work was supported by the University of California, Riverside (BCOE, Regents' Faculty Fellowship).

References

[1] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *IEEE Intl. Conf. on Computer Vision*, vol. 2, Nice, France, Oct. 2003, pp. 1403–1410.

[2] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *IEEE Intl. Conf. on Robotics and Automation*, Roma, Italy, Apr. 2007, pp. 3565–3572.

[3] K. Konolige, M. Agrawal, and J. Sola, "Large-scale visual odometry for rough terrain," in *Intl. Symposium on Research in Robotics*, Hiroshima, Japan, Nov. 26-29 2007.

[4] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry for ground vehicle applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, Jan. 2006.

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Eighteenth National Conf. on Artificial Intelligence*, July/Aug. 2002, pp. 593–598.

[6] F. Chenavier and J. L. Crowley, "Position estimation for a mobile robot using vision and odometry," in *IEEE Intl. Conf. on Robotics and Automation*, Nice, France, May 1992, pp. 2588–2593.

[7] A. Wu, E. Johnson, and A. Proctor, "Vision-aided inertial navigation for flight control," San Francisco, CA, Aug. 2005.

[8] N. Trawny, A. I. Mourikis, S. I. Roumeliotis, A. E. Johnson, and J. Montgomery, "Vision-aided inertial navigation for pin-point landing using observations of mapped landmarks," *Journal of Field Robotics*, vol. 24, no. 5, pp. 357–378, May 2007.

[9] J. Montiel, J. Civera, and A. Davison, "Unified inverse depth parametrization for monocular SLAM," in *Proceedings of Robotics: Science and Systems*, Philadelphia, PA, Aug. 16-19 2006, pp. 81–88.

[10] M. Pupilli and A. Calway, "Real-time camera tracking using a particle filter," in *British Machine Vision Conf.*, Oxford Brookes University, Oxford, UK, Sept. 2005, pp. 519–528.

[11] E. Eade and T. Drummond, "Scalable monocular SLAM," in *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, vol. 1, New York, New York, USA, June 2006, pp. 469–476.

[12] D. S. Bayard and P. B. Brugarolas, "An estimation algorithm for vision-based exploration of small bodies in space," in *American Control Conf.*, vol. 7, Portland, Oregon, USA, June 2005, pp. 4589–4595.

[13] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters for view-based SLAM," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1100–1114, Dec. 2006.

[14] L. H. Matthies, "Dynamic stereo vision," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, 1989.

[15] P. F. McLauchlan, "The variable state dimension filter applied to surface-based structure from motion," Centre for Vision, Speech and Signal Processing, University of Surrey, Tech. Rep., 1999.

[16] S. Julier, J. Uhlmann, and H. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," vol. 45, no. 3, pp. 477–482, 2000.

[17] K. Ito and K. Xiong, "Gaussian filters for nonlinear filtering problems," *IEEE Transactions on Automatic Control*, vol. 45, no. 5, pp. 910–927, May 2000.

[18] B. Triggs, P. McLauchlan, R. Hartley, and Fitzgibbon, "Bundle adjustment — a modern synthesis," in *Vision Algorithms: Theory and Practice*. Springer Verlag, 2000, pp. 298–375.

[19] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellised particle filters by adaptive proposals and selective resampling," in *IEEE Intl. Conf. on Robotics and Automation*, Barcelona, Spain, Apr. 2005, pp. 2432–2437.

[20] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 590–599, 1999.

[21] J.-Y. Bouguet, "Camera calibration toolbox for Matlab," http://www.vision.caltech.edu/bouguetj/calib_doc/, June 2008.

[22] J. Shi and C. Tomasi, "Good features to track," in *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, Seattle, Washington, USA, June 1994, pp. 593–600.

[23] S. Birchfield, "KLT: Kanade-Lucas-Tomasi feature tracker," http://www.ces.clemson.edu/~stb/klt/installation.html, Aug. 2007.