Comparing the EKF and FastSLAM solutions to the problem of monocular Simultaneous Localization and Mapping.

Gert Kootstra and Sjoerd de Jong and Daniël Wedema
July 24, 2009

Abstract

This paper presents a comparison of the extended Kalman filter (EKF-SLAM) and FastSLAM algorithms, the two most popular solutions to the simultaneous localization and mapping problem (SLAM). We focus strictly on the class of monocular indoor mobile robots. Because the extended Kalman filter only maintains one solution, it is known to be very fragile to incorrect data-associations. FastSLAM, on the other hand, has it’s background in particle filtering, in which it tracks multiple hypotheses at the same time, which enable it do be robust under incorrect data-associations. Most papers comparing the two algorithms do this by means of simulation, or on real-world data with poor ground truth, making it difficult to estimate the performance of the filter. We however compare both algorithms on real-world data with high-precision ground truth available. The experiments show that the extended Kalman filter suffers significantly from incorrect data-associations. FastSLAM does not suffer significantly from the incorrect data-associations and is thus a robust filter in situations where the data-association problem is very hard. We also show that, despite the better performance, FastSLAM is less consistent compared to EKF-SLAM.

1 Introduction

For a robot to be truly autonomous, it will need some way to create a map of its surrounding when it is placed at an unknown location in an unknown surrounding. Simultaneous localization and mapping (SLAM) [Bailey and Durrant-Whyte, 2006b] is the problem of trying to locate where a robot is and at the same time building a map based on a series of actions and observations. The two main solutions to this problem are the extended Kalman filter (EKF-SLAM) approach which maintains one hypothesis of the solution and a particle filter approach which tracks multiple hypotheses. In this article, we will look at the differences between these solutions and compare their performance in different situations.

One of the difficult aspects of SLAM is that the robot starts out with no information about its location, or about his environment. To accurately map the observations, a robot must know its location. Any error in the location estimation will produce errors in the map. However, to estimate its own location, a robot needs an accurate map. Any error in the map will produce errors in localization. To solve the SLAM problem, the robot will need to bootstrap itself out of this situation.

In very early work on this topic it was believed that the landmark and robot locations could be propagated independently of one another. Work by Smith et al. [1988] however showed that errors in location estimates were necessarily correlated. These errors are correlated because each landmark position estimate incorporates the same error in vehicle position estimate. This meant that the location estimates could not be propagated independently of each other and the correlations between all the errors in location estimates had to be maintained to find a good solution. When these correlations were not maintained the filter would be overconfident in the information it has and the filter would create an erroneous map and an inconsistent pose estimate [Castellanos et al., 1997].
This insight finally led to the use of an extended Kalman filter (EKF) based approach to solve the SLAM problem, in which the solution to the problem is represented as a Gaussian distribution over the state variables. The Kalman filter stores the correlations between all location estimate errors. This way when the robot observes a landmark and updates its location, it will also know more about the location of the other landmarks and robot pose because their errors are correlated. Dissanayake et al. [2001] proved that in the limit, as more and more observations are made, the errors between the landmark and robot location estimates become fully correlated. This means that given the exact location of one of the landmarks, the locations of all other landmarks are also exactly known. This in turn implies that the mapping and localization error is only dependent on initial uncertainty in the robot pose.

Because the EKF approach to SLAM stores the correlation between every pair of landmarks, the space and time required in each iteration scales quadratically with the number of landmarks in the map. Another problem in EKF-SLAM is the use of a Gaussian distribution as a solution. The Gaussian assumption, although practical in calculations, is not true in reality. This makes real world application for large scale environments infeasible. Much work has been done to reduce the time and space complexity, for a short overview see [Bailey and Durrant-Whyte, 2006a].

Another approach to SLAM is the use of a Monte-Carlo method. In this method a set of samples (particles) are randomly generated to represent possible solutions of the SLAM problem. The solution space is too large to sample directly, so the technique of Rao-Blackwellisation [Blackwell, 1947] is used to reduce the complexity. Here, only a subset of the state is sampled, the other variables are estimated analytically [Doucet et al.]. When possible robot poses are sampled, each sample assumes absolute certainty of the robot pose and thus has to estimate only the landmark locations. This problem is essentially a mapping problem which can be solved for each particle with independent Kalman filters for the landmark estimates of that particle. Because the landmark estimates are independent of each other, the problem scales linearly in space and time with respect to the number of landmarks stored by each particle.

The FastSLAM filter introduced in [Montemerlo et al., 2002] is an application of a Rao-Blackwellised particle filter. The space required in this filter grows linearly in the number of landmarks and in the number of particles. The time complexity grows linearly in the number of landmarks and logarithmic in the number of particles. In [Montemerlo et al., 2003], an improved version of FastSLAM is presented and convergence for linear SLAM problems when using one particle is proven. This is interesting because FastSLAM with one particle is almost identical to EKF-SLAM while ignoring cross-correlations [Bailey et al., 2006]. In Castellanos et al. [1997] EKF-SLAM with and without cross-correlations, except that EKF-SLAM with cross-correlation is consistent while EKF-SLAM without cross-correlations is inconsistent.

To date there has been little direct comparison of both algorithms on real-world data [Montemerlo and Thrun, 2003]. One reason for the lack of direct comparison is that it is hard to get good-quality ground truth information to measure the performance of the algorithms. Another reason is that it is difficult to accurately control the environment. For these reasons, both filters are mainly tested in simulations where everything can be overseen. We however have very accurate ground truth of the real robot location, which enables us to measure the performance of the localization part of both algorithms very accurately.

In Castellanos et al. [2004] an EKF-SLAM algorithm is tested in simulations. In this work Castellanos et al. show that the filter is inconsistent because of the linearisation errors introduces by approximating the non-linear system by a linear one. These inconsistencies show up before any computational problems arise. Bailey et al. [2006] show in simulations that FastSLAM is an inconsistent filter as well. They argue that this is caused by resampling the particles. Each time a particle is not se-
lected for the next time-step, an entire pose history with associated map is deleted, causing the filter to lose information. Bekris et al. [2006] provides an extensive comparison of a wide range of SLAM algorithms, including EKF-SLAM and FastSLAM. They show that FastSLAM has a greater percentage of successful runs than EKF-SLAM. Furthermore their results show that the performance of EKF-SLAM deteriorates with a higher landmark density. The performance of FastSLAM is unchanged when the landmark density changes.

One of the major drawbacks of the extended Kalman filter is its inability to deal with ambiguous observations. In this article we will investigate this inability more accurately. In section 2 we will explain the details of both algorithms. The set-up of our experiments is shown in section 3. In section 4 we will show that the performance of the extended Kalman filter is affected significantly by highly ambiguous observations, while the FastSLAM algorithm is not significantly affected when confronted with the same data. Finally in section 5 we will discuss these results.

2 Methods

To solve the full SLAM problem, the posterior distribution $P(X^t, M|Z^t, U^t, x_0)$ needs to be computed. Here $X^t = \{x_0, \cdots, x_t\}$, $M = \{m_0, \cdots, m_k\}$, $Z^t = \{z_1, \cdots, z_t\}$ and $U^t = \{u_1, \cdots, u_t\}$ are the collections of the robot poses $x_t = (x_t, y_t, \phi_t)^T$, the landmark locations $m_t = (x_t, y_t)^T$, the observations $z_t = (d_t, \beta_t)^T$ given as (range, bearing) and the robot actions $u_t = (v_t, \Delta \phi_t)^T$ respectively. All location coordinates are stored in global coordinates, relative to the initial robot pose. The observations and robot actions are relative to the current robot pose.

Both algorithms apply the Markov assumptions to compute this posterior distribution recursively. The first assumption is that the current robot pose only depends on the previous pose. The new robot pose is then predicted according to the motion model $P(x_{t|t-1}, u_t)$. The second assumption is that the observation only depends on the current robot pose and map. Using this assumption, a correction is applied to the robot pose and landmarks by using the observation model $P(z_t|x_t, M)$.

In section 2.1 we will describe the EKF-SLAM algorithm. In section 2.2 the FastSLAM algorithm is explained. In section 2.3 we will describe how we use a monocular camera to estimate the landmark locations.

2.1 Extended Kalman Filter

The basis for the extended Kalman filter is to describe the joint posterior distribution $P(x_t, M|Z^t, U^t, x_0)$ as a Gaussian distribution over the variables $x_t$ and $M$. To do this the estimated mean $(\hat{x}_t, \hat{M})$ and the covariance $P_t = (P_{xx} P_{xm} P_{xm})$ are maintained. Here $P_{xx}$ is the error covariance matrix over the robot pose, $P_{xm} = P_{mx}^T$ is the cross-correlation between the robot pose and landmarks and $P_{mm}$ contains the error covariance matrices and the cross-correlations of the individual landmarks.

The filter recursively estimates the mean and covariance in two steps. In the predict step the pose is updated using the odometry information, in the correct step the visual information is used to correct the estimated pose and map.

2.1.1 Prediction step

To predict the next robot pose using the odometry, a motion model of the behaviour of the robot is required. This model predicts where the robot will be, given the current robot pose and a control input. The motion model is described in the form

$$P(x_t|x_{t-1}, u_{t-1}) \leftrightarrow x_t = f(x_{t-1}, u_t) + w$$

where $f(\cdot)$ models the behaviour of the robot and $w$ is a Gaussian motion disturbance with zero-mean and covariance $Q$. We assume that the covariance $Q$ is diagonal and constant over time.

Using this model, the mean estimate and covariance can be predicted according to

$$P(x_t, M|Z_{t-1}, U_t, x_0) =$$

$$\int P(x_t|x_{t-1}, u_t)P(x_{t-1}, M|Z_{t-1}, U_{t-1}, x_0)dx_{t-1}$$
Because the system is a Gaussian distribution, the resulting distribution is another Gaussian which can be written as

\[
\begin{align*}
\begin{pmatrix} \hat{x}_t \\ M \end{pmatrix} &= f(\hat{x}_{t-1}, u_t) \\
\begin{pmatrix} P_t^- \end{pmatrix} &= \nabla f P_{t-1} \nabla f^T + Q
\end{align*}
\]

Here \(\hat{x}_t\) and \(P_t^-\) are the best guesses up to time \(t\), without using observation \(z_t\). \(\nabla f\) is the Jacobian of \(f\) evaluated at the estimate \(\hat{x}_{t-1}\). The Jacobian is necessary to linearise the non-linear function \(f\). The motion model and Jacobian are given as

\[
\begin{align*}
 f(\hat{x}_{t-1}, u_t) &= \begin{pmatrix} \hat{x}_{t-1} + v_1 \Delta t \cos(\hat{\phi}_{t-1} + \Delta \phi_t) \\ \hat{y}_{t-1} + v_2 \Delta t \sin(\hat{\phi}_{t-1} + \Delta \phi_t) \\ \hat{\phi}_{t-1} + \Delta \phi_t \end{pmatrix} \\
\nabla f &= \begin{pmatrix} \frac{\partial f}{\partial \hat{x}_{t-1}, M} \end{pmatrix} = \begin{pmatrix} F_x & 0 \\ 0 & 1 \end{pmatrix} \\
F_x &= \begin{pmatrix} 1 & 0 & -\sin(\hat{\phi}_{t-1} + \Delta \phi_t) \\ 0 & 1 & \cos(\hat{\phi}_{t-1} + \Delta \phi_t) \\ 0 & 0 & 1 \end{pmatrix}
\end{align*}
\]

After applying the predict step, the state estimate becomes more uncertain, i.e. the Shannon entropy of the covariance matrix \(P_t^-\) will be bigger than the Shannon entropy of the covariance matrix of the previous time-step \(P_{t-1}\).

### 2.1.2 Correction step

In the second step, the observation model predicts the range and bearing to landmarks given the current robot pose and the map. This prediction is compared to the real observation to correct the state estimate. The observation model is given as

\[
P(z_t | x_t, M) \leftrightarrow z_t = h(x_t, M) + u_t
\]

where \(h(\cdot)\) models the behaviour of the observation device and \(u_t\) is a Gaussian observation noise with zero-mean and covariance \(R_t\). This covariance is a diagonal matrix and depends on the accuracy of the observation (see section 2.3 for the calculation of \(R_t\)).

Using this model, the estimated pose and map can be corrected according to

\[
P(x_t, M | Z_t, U_t, x_0) = \frac{P(z_t | x_t, M) P(x_t, M | Z_{t-1}, U_t, x_0)}{P(z_t | Z_{t-1}, U_t)}
\]

which, under the Gaussian observation model, results in a new Gaussian distribution with mean and covariance

\[
\begin{align*}
\begin{pmatrix} \hat{x}_t \\ M \end{pmatrix} &= \begin{pmatrix} \hat{x}_t^- \\ M \end{pmatrix} + K \epsilon \\
P_t &= (I - K \nabla h) P_t^-
\end{align*}
\]

where

\[
\begin{align*}
\epsilon &= z_t - h(\hat{x}_t^-, M) \\
K &= P_t^- \nabla h^T S^{-1} \\
S &= \nabla h P_t^- \nabla h^T + R_t
\end{align*}
\]

Here \(\nabla h\) is the Jacobian of \(h\) with respect to the complete state, evaluated at the pose \(\hat{x}_{t-1}\) and landmark \(\hat{m}_i\). The observation model and Jacobian for observing landmark \(\hat{m}_i\) are given as

\[
\begin{align*}
h(\hat{x}_t, M) &= \begin{pmatrix} \Delta_i \\ \tan^{-1} \left( \frac{\Delta y_i}{\Delta x_i} \right) \end{pmatrix} \\
\nabla h &= \begin{pmatrix} \frac{\partial h}{\partial \hat{x}_t, M} \end{pmatrix} = \begin{pmatrix} -H_i, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \end{pmatrix} \\
H_i &= \begin{pmatrix} \frac{\delta x_i}{\Delta x_i} & \frac{\delta y_i}{\Delta x_i} \\ \frac{\delta x_i}{\Delta y_i} & \frac{\delta y_i}{\Delta y_i} \end{pmatrix}
\end{align*}
\]

where \(\Delta x_i = \hat{x}_i - \hat{x}_i\), \(\Delta y_i = \hat{y}_i - \hat{y}_i\) and \(\Delta_i = \sqrt{(\Delta x_i)^2 + (\Delta y_i)^2}\). Here the index \(i\) is used as index for landmark \(\hat{m}_i\). For one observation, the Jacobian \(\nabla h\) is a \(2 \times (3 + 2N)\) matrix, where \(N\) is the number of landmarks already in the map. This matrix is mostly zero, except for the first 3 columns (which describe the influence on the robot pose) and the columns corresponding to the observed landmark \(\hat{m}_i\). When there are multiple observed landmarks at one time, more rows are added to the Jacobian.

After integrating the observational data with the use of the observation model, the state estimate becomes less uncertain, i.e. the Shannon entropy of the
covariance matrix $P_t$ is smaller than the Shannon entropy of the covariance matrix $P_t^{-}$.

When one of the observations does not match the landmarks in the map, it is added as a new landmark. The state is extended by adding the mean of this landmark. The mean is given by the inverse of the observation model $m_i = h^{-1}(x_t, z_i)$. The covariance is updated according to

$$P_t^+ = \nabla h_x^{-1} P_t (\nabla h_x^{-1})^T + \nabla h_m^{-1} R_t (\nabla h_m^{-1})^T$$

Here $\nabla h_x^{-1}$ and $\nabla h_m^{-1}$ are the Jacobian of the inverse observation model with respect to the robot pose and landmark respectively. For one new landmark, the new covariance matrix $P_t^+$ extends by two rows and two columns.

The Jacobians are given by

$$\nabla h_x^{-1} = \frac{\partial h^{-1}}{\partial x_t} = \begin{pmatrix} I & H_x^{-1} \end{pmatrix}$$

$$H_x^{-1} = \begin{pmatrix} 1 & 0 & -d_t \sin(\phi_t + \beta_t) & 0 & \cdots & 0 \\ 0 & 1 & d_t \cos(\phi_t + \beta_t) & 0 & \cdots & 0 \end{pmatrix}$$

and

$$\nabla h_m^{-1} = \frac{\partial h^{-1}}{\partial m_i} = \begin{pmatrix} 0 \\ H_m^{-1} \end{pmatrix}$$

$$H_m^{-1} = \begin{pmatrix} \cos(\phi_i + \beta_i) & -d_t \sin(\phi_i + \beta_i) \\ \sin(\phi_i + \beta_i) & d_t \cos(\phi_i + \beta_i) \end{pmatrix}$$

Here the identity matrix $I$ is of size $(3 + 2N) \times (3 + 2N)$. The zero matrix is of size $(3 + 2N) \times 2$.

2.2 FastSLAM

FastSLAM [Montemerlo et al., 2002] has its basis in recursive Monte Carlo sampling, or particle filtering, where the distribution is represented by a set of samples that approximate the posterior distribution. For $N$ landmarks, the dimensionality of the SLAM state space is $3 + 2N$. The number of particles needed to explore this space grows exponentially with the dimensionality. Direct sampling of this space becomes a problem with too many landmarks. The method of Rao-Blackwellization [Doucet et al., Robert et al., 2005] reduces this complexity by sampling a subset of the distribution and marginalizing out the remaining ones.

FastSLAM is an implementation of a Rao-Blackwellized particle filter. In FastSLAM the posterior distribution is factored as

$$P(X_t, M|U_t, Z_t, x_0) =$$

$$P(X_t|U_t, Z_t, x_0) \times \prod_{k=0}^M P(m_k|X_t, Z_t)$$

The distribution $P(X_t|U_t, Z_t, x_0)$ is sampled by individual particles, and each distribution $P(m_k|X_t, Z_t)$ is calculated analytically by an EKF. Because the observations only depend on the map and the robot pose, the estimations of the landmarks become conditionally independent when the path of the robot is given. In FastSLAM this means that each EKF is only of size $2 \times 2$. This implies that FastSLAM only needs to store the diagonal of the covariance matrix $P_t$ which is stored in EKF-SLAM, making the SLAM solution linear in the number of landmarks (with a constant depending on the number of particles). This is a big gain compared to the quadratic complexity in EKF-SLAM.

The FastSLAM algorithm represents the posterior as the set $S = \{x_t^{[k]}, M^{[k]}, w^{[k]}\}_k$. Each particle $k$ stores the estimated pose $x_t^{[k]}$ and a map $M^{[k]} = \{m_i^{[k]}, \Sigma_i^{[k]}\}_{i}$ with the mean $m_i^{[k]}$ and covariance $\Sigma_i^{[k]}$ of each landmark $i$. The weight $w^{[k]}$ is the likelihood of the particle, as explained in section 2.2.3.

The algorithm assumes that the set $S_{t-1}$ is distributed according to the posterior $P(x_{t-1}, M|U^{t-1}, Z^{t-1}, x_0)$ and calculates the new set $S_t$ recursively in three steps. First the pose of each particle is predicted using the odometry information. In the correction step the visual information is used to correct the map and recalculate the likelihood of each particle. Finally a resample step is necessary since only a finite number of particles is used to represent the target distribution.

2.2.1 Prediction step

The predict step is done by sampling from the motion model. For each particle the pose is predicted
between the proposal distribution and the target distribution of their weight. The weights reflect the difference between the proposal distribution with probability proportional to \( \phi_{t-1} + \Delta \phi \delta \phi_a \)

\[
\begin{pmatrix}
x_{t-1}^{[k]} + v_t \delta v \cos(\phi_{t-1} + \Delta \phi \delta \phi_a) \\
y_{t-1} + v_t \delta v \sin(\phi_{t-1} + \Delta \phi \delta \phi_a) \\
\phi_{t-1} + \Delta \phi \delta \phi_p
\end{pmatrix}
\]

Here \( \delta v \), \( \delta \phi_a \) and \( \delta \phi_p \) are three independent pseudo-random values from a Gaussian distribution with mean 1 and constant variance.

This way a temporary particle set is created which is distributed according to \( P(x_t, M|U_t, Z^{t-1}, x_0) \) which is commonly referred to as the proposal distribution of particle filtering.

### 2.2.2 Correction step

In the correction step the map is corrected using the newest observation \( z_t \). This is done using a regular EKF correction step. For every observed landmark \( i \) and particle \( k \) we use the following correction equations:

\[
\begin{align*}
\mathbf{m}_{t,i}^{[k]} &= \mathbf{m}_{t-1,i}^{[k]} + K_i^{[k]} \mathbf{e}_i^{[k]} \\
\Sigma_{i,t}^{[k]} &= (I - K_i^{[k]} H_i) \Sigma_{i,t-1}^{[k]}
\end{align*}
\]

where

\[
\begin{align*}
\mathbf{e}_i^{[k]} &= \mathbf{z}_i - \mathbf{h}(\mathbf{x}_i^{[k]}, \mathbf{m}_{t-1,i}^{[k]}) \\
K_i^{[k]} &= \Sigma_{i,t-1}^{[k]} H_i^{[k]T} S_i^{[k]-1} \\
S_i^{[k]} &= H_i^{[k]} \Sigma_{i,t-1}^{[k]} H_i^{[k]T} + R_t
\end{align*}
\]

Here \( H_i^{[k]} \) is the Jacobian as defined in section 2.1.2 for landmark \( i \) in the map of particle \( k \).

### 2.2.3 Resample step

The pose and map are simultaneously updated in the resample step which resamples particles from the proposal distribution with probability proportional to their weight. The weights reflect the difference between the proposal distribution and the target distribution. They are defined to be

\[
\begin{align*}
\mathbf{w}_i^{[k]} &= \frac{\text{target distribution}}{\text{proposal distribution}} = \frac{P(x_t^{[k]}|U_t, Z^t, x_0)}{P(x_t^{[k]}|U_t, Z^{t-1}, x_0)}
\end{align*}
\]

As shown in Montemerlo et al. [2002] this can be approximated using the Markov and EKF assumptions by \( \mathbf{w}_i^{[k]} \approx \int P(z_i|\mathbf{m}_{t,i}^{[k]}, x_t^{[k]}) P(\mathbf{m}_i) d\mathbf{m}_i \) which we calculate for each observation and particle in closed form as

\[
\mathbf{w}_i^{[k]} = |2\pi S_i^{[k]}|^{-\frac{1}{2}} \exp \left( -\frac{\mathbf{e}_i^{[k]T} S_i^{[k]-1} \mathbf{e}_i^{[k]}}{2} \right)
\]

where \( S_i^{[k]} \) is the measurement uncertainty and \( \mathbf{e}_i^{[k]} \) is the measurement error as defined in the previous section. We define the total weight for each particle to be \( \mathbf{w}^{[k]} = \sum_i \mathbf{w}_i^{[k]} \).

When to resample is still an open question. Resampling each timestep might cause the filter to throw away potentially good particles too soon. An often used technique is to resample only when the number of particles with a significant weight is below a threshold [Grisetti et al., 2007]. We however have chosen to resample only when the previous observation of a currently observed landmark is more than 150 time-steps ago. By doing this we make sure resampling only happens when the robot returns to a previously visited location, and not when a landmark from the previous time-step is observed again.

When an observation does not match any observation in the map, it is added to the map. The mean of this landmark is given by the inverse of the observation model \( \mathbf{m}_i^{[k]} = h^{-1}(x_i^{[k]}, z_i) \). The covariance is initialized as \( \Sigma_i^{[k]} = H_m^{-1} R_l(H_m^{-1})^T \).

### 2.3 Monocular feature estimation

The observations \( z_t = \{d_i, \beta_i\} \) are given as (range, bearing) pairs. To do this, depth needs to be estimated from the sensor. In our case we are using a single camera. To estimate the distance to a landmark, a constant size image buffer as described in [de Jong, 2008] is used to store pictures from the last \( T \) time-steps. Each time-step the most recent picture is added, and the oldest is removed. The use of this buffer has two advantages. By tracking robust features over time in the buffer, depth information can be extracted with triangulation. The second advantage is that the buffer allows for the selection of the most robust features.
Figure 1: Examples of camera images. The dots in the image are observations coming from the visual buffer. The yellow numbers are observations of new landmarks which are added to the map, the blue numbers are observations of known landmarks which are used to update the map and robot pose.

Each picture is processed to find invariant features based on the Scale Invariant Feature Transform (SIFT) [Lowe, 1999]. This results in a set of invariant points (see Figure 1), each described by an 128-dimensional feature vector $f$. The feature vectors from the most recent picture are matched against feature vectors in the other pictures in the buffer using an euclidean distance measure.

Two criteria are used to find a positive match in each picture. First the ratio between the best match and second best match has to be below a buffer distance ratio threshold which ensures that only the most discriminative features are selected. Another criterion is that the euclidean distance has to be below a buffer distance threshold which ensures that the feature vectors are similar enough.

A SIFT feature is considered an observation of a landmark when it is seen a minimum number of times in the buffer. The different matches can then be used to triangulate the position of the landmark. The observation is extended as $z_t = \{d, \beta, f\}$ to be able to perform data association. Together with this position estimate a covariance $R_t = \begin{pmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\beta^2 \end{pmatrix}$ of the observation is calculated from the different matches in the buffer.

To complete the data association, we also extend the landmarks as $m_i = \{x_i, y_i, f_i\}$ and use the features from the observations to find similar looking landmarks using the euclidean distance measure between features. Here again two thresholds are used (the association distance threshold and association distance ratio threshold) to find the best match.

Finally the Mahalanobis distance between expected landmark location and observed landmark location is used. This way we not only match on visual information, but also on the expected location of the observed landmark. When the Mahalanobis distance exceeds the max mahalanobis threshold, the observation is ignored. This distance is calculated as

$$d(m_i, h^{-1}(x_t, z_t), C_t) = \sqrt{\epsilon^T C_t^{-1} \epsilon}$$

Here the error is given by $\epsilon = m_i - h^{-1}(x_t, z_t)$. The covariance $C_t$ describes the total uncertainty of the landmark estimate. In the case of the Kalman filter, the total uncertainty is determined by the uncertainty in robot pose and uncertainty in the observation. So the covariance is calculated as $C_t = H^{-1} P_{xx}(H_x^{-1})^T + H^{-1} R_t(H_m^{-1})^T$. In the particle filter the Mahalanobis distance check is done on a per particle basis, which means there is no pose uncertainty and the Mahalanobis distance is calculated with $C_t = H^{-1} R_t(H_m^{-1})^T$.

### 3 Experimental setup

For this experiment, a Pioneer 2 DX mobile robot is driven manually in a closed loop route in an office-like environment. One loop is 34 meter long. The robot is equipped with a single camera, mounted on top facing forward. At regular intervals of $\Delta t = 0.8$ sec, camera images are stored along with the odometric information from the wheel-encoders. The odometric
information is used to calculate the robot action $u_t = \{v_t, \Delta \phi_t\}$.

Data is collected in datasets and later used as input for the algorithms. Because we collected the data in datasets in advance, we could test both algorithms on exactly the same input. Each run consists of four sequential loops. In total 6 runs were used as data in the dataset, each run with a different starting position or driving direction (clockwise or counter-clockwise).

The ground truth was created using marked points every meter along the driven loop shown in figure 2. Every time the robot passed these marked points, the deviation from the centre together with the odometric robot pose was logged manually by a key-press.

To see how both algorithms deal with highly ambiguous data-associations, we created two sets of settings for the visual buffer. In the first set (reliable data-association) we chose optimal settings for the filters. The optimal settings were taken from de Jong [2008].

In the second set (unreliable data-association) we increased the Mahalanobis distance, so the information about location becomes less accurate and the filter has to rely more on visual information only. Together with this we degraded the quality of the visual information by setting the distance and ratio thresholds to higher values. By increasing the ratio thresholds, less discriminative features are added to the map, increasing the ambiguity. Increasing the distance thresholds will increase the probability of a match, causing more false matches. See Table 1 for the exact settings.

In both settings we set the visual buffer size to 8 images, which corresponds to a buffer of 6.4 seconds. A feature is considered an observation when it is seen at least 4 times in the visual buffer.

<table>
<thead>
<tr>
<th>threshold variable</th>
<th>reliable</th>
<th>unreliable</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer distance ratio</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>buffer distance</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td>association distance ratio</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>association distance</td>
<td>120</td>
<td>250</td>
</tr>
<tr>
<td>max mahalanobis</td>
<td>5</td>
<td>500</td>
</tr>
</tbody>
</table>

On average 272.4 SIFT feature points are extracted from each picture in our dataset. When the reliable settings are applied to the visual buffer, it generates on average 1.4 observations for each picture. With the unreliable settings, the visual buffer generates on average 2.5 observations for each picture.

Every time the estimated robot pose passes the markpoints, the filter gave the estimated robot pose as output. From this estimate and the true position at these markpoints, we can accurately calculate the estimation error. To measure the performance of the filter, we drove the robot 4 times through the same loop. The first 3 rounds were used to let the map converge. We used lap 4 to measure the error of the filter by measuring the distance from the estimated position to the true position.

Another interesting feature of filters is consistency. A filter is consistent if the calculated covariance represents the actual estimation error. Because we have an accurate ground truth of the robot location, we can carry out a statistical test for filter consistency. We calculate the normalized estimation error squared (NEES) as

$$D^2 = (x - \hat{x})^T P^{-1} (x - \hat{x})$$

For this test we only use the $(x, y)$ coordinates of the robot location, so the covariance matrix $P$ is of size $2 \times 2$. When a filter is consistent, the value $D^2$ has a
$\chi^2_r$ density with $r = \text{dim}(\mathbf{x})$ degrees of freedom. We check consistency according to

$$D^2 \leq \chi^2_{r, 1-\alpha}$$

with significance level $\alpha = 0.05$ and $r = 2$, the threshold for consistency is $D^2 \leq 5.99$.

In all experiments we use FastSLAM with 1000 particles. To see how the number of particles influence the performance of the filter, we varied the number of particles between 1 and 10,000 particles.

In the case of the extended Kalman filter, the output is deterministic given the settings and input. This means we only needed to run the filter once on each run and set of settings to know the exact performance of the filter. The FastSLAM filter on the other hand is stochastic, so we executed each run and set of settings 15 times, and averaged the errors to estimate the performance of each run.

In all the experiments we have used the control noise $Q = \begin{pmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_{\phi}^2 \end{pmatrix}$, with $\sigma_v = 1.8$ m/s and $\sigma_{\phi} = 0.9^\circ$ in the prediction step of the extended Kalman filter. In FastSLAM we get comparable motion uncertainty when we generate $\delta v$, $\delta \phi_a$, and $\delta \phi_p$ according to $\sim N(1, 0.06)$.

4 Results

The performance of the filters on all runs are shown in Figure 3. The performance is measured as the average error in the last loop of each run. This plot shows that both filters improve the result compared to the odometry. From this plot we can see that the performance of FastSLAM is better, with both the reliable and unreliable visual buffer settings. Another interesting fact is that FastSLAM does not seem to suffer very much from the unreliable visual buffer, whereas the performance of EKF decreases drastically when the visual buffer is set to unreliable.

To make this more precise, we performed a paired T-test to test whether both filters are significantly better compared to the odometry. The extended Kalman filter and FastSLAM both significantly improve the result compared to the odometry, with reliable (respectively $p < 10^{-20}$ and $p < 10^{-22}$) and unreliable (respectively $p < 10^{-16}$ and $p < 10^{-24}$) visual buffer settings. Another paired T-test showed that FastSLAM significantly outperformed the extended Kalman filter on both the reliable ($p < 10^{-13}$) and unreliable ($p < 10^{-17}$) settings.

Two final paired T-tests were performed to see whether the unreliable visual input had a significant effect on the results of each filter. This is the case for the extended Kalman filter. The filter becomes significantly worse ($p < 10^{-14}$) when wrong data-associations are incorporated. Although the performance of FastSLAM decreases slightly in our experiments, FastSLAM does not significantly suffer ($p = 0.1$) from wrong data-associations.

In Figure 4 we show an example of a run of the FastSLAM filter. The top-right plot shows the original odometric information after four loops. The robot starts at position (0, 0), and drives clockwise. Because of accumulated errors, the odometric estimate has drifted significantly from the true position (top-left). The bottom plots show the filtered robot pose after integrating both visual and odometric information.
The drifting of the estimated location is resolved by both filters, but both solutions are slightly rotated compared to the true solution. This rotation is caused by the initial error in the odometry in the first lap. In the second lap, the filter reobserves landmarks seen in the previous lap and starts to follow the map of the first lap, which is already slightly rotated.

A comparison of the evolution of the errors of both filters in a single run is shown in figure 5. Here the error is measured as the distance from the estimate position to the true position in millimetres. As seen in Figure 4 the odometry error increases each lap. In the first lap the error of both filters follow the error of the odometry. After the first lap the robot reobserves landmarks from the previous lap, which enable it to update its position and map, which cause the filter error to get lower. The maximum error in each subsequent lap is bounded.

When the visual buffer is set to unreliable (dashed red line) less reliable observations enter the map. Because of the single hypothesis tracking, EKF can adjust its pose over great distances. This can have destructive consequences when EKF makes a wrong data-association, as seen in lap 2. In the FastSLAM algorithm these wrong data-associations are not as destructive, because robot poses are updated by selecting them from the proposal distribution, which is represented by a particle cloud.

In figure 6 an example run is shown where the consistency of both filters is measured. Although the performance for the reliable visual buffer settings are comparable (Figure 5), EKF seems to be more consistent than FastSLAM. In this run EKF is 87.5% of the time below the consistency threshold, while FastSLAM is only 41.1% of the time consistent. With the unreliable visual buffer settings (dashed line), EKF is still consistent more than half the time (59.8%) which is much better than FastSLAM (28.6%).

Finally in Figure 7 we show the results from all the runs when we vary the number of particles in FastSLAM. The average error of EKF is shown as the horizontal dashed line. This plot shows that the error of FastSLAM decreases asymptotically as we increase the number of particles. Note that for as little as 10 particles FastSLAM is almost as good as EKF. This suggests that for these small indoor en-
environments, somewhere between 10 and 100 particles will be enough to equal the performance of EKF.

5 Discussion

In this article we have compared the two most popular approaches to solve SLAM. We have given an example of how to apply this solution to a monocular indoor SLAM problem. Because we have very accurate ground truth information, we can also accurately compare and test both filters.

We have shown that both the extended Kalman filter and FastSLAM can improve the odometric information given by the robot. Secondly we have shown that FastSLAM outperforms the extended Kalman filter when compared on estimation error. Although FastSLAM has a lower error, our experiments suggest that it is less consistent than the extended Kalman filter. This can become problematic in the long run, because inconsistency can lead to divergence of the filter.

Finally we have shown that FastSLAM is more robust than the extended Kalman filter when confronted with ambiguous data association. Even though the extended Kalman filter still improves the odometry, it is significantly affected by the wrong associations. FastSLAM can deal with wrong associations more robustly by maintaining multiple independent hypotheses, each making it’s own data associations. The particles with the wrong data association will be removed in subsequent resample steps and replaced by more likely particles. Another reason for the robustness of FastSLAM compared to EKF is that the pose estimate in EKF can jump large distances, while in FastSLAM it is bounded by the particle cloud.

These experiments suggest that when data-associations are unreliable, FastSLAM is the better solution. For longer runs with reliable data-associations EKF is a better solution as it seems to be more consistent. Future experiments will have to examine the consistency and convergence of longer runs, as the inconsistency of FastSLAM as shown here can lead to divergence of the filter.
References


