Particle Filters for Mobile Robot Localization:

A Practitioner's Tutorial

Particle Filters for Mobile Robot Localization: A Practitioner's Tutorial

Ioannis M. Rekleitis
Centre for Intelligent Machines, McGill University,
3480 University St., Montreal, Québec, CANADA H3A 2A7

Published by Y-one-D books.
http://Y1D.com
Montreal, QC Canada

ISBN 978-0-9809915-3-6

Library and Archives Canada Cataloguing in Publication
NAME(S) : Rekleitis, Ioannis
TITLE(S) : Particle Filters for Mobile Robot Localization: A Practitioner's Tutorial / Ioannis Rekleitis
NUMBERS: Canadiana:
CLASSIFICATION: SUBJECTS: Mobile Robotics

1. Mobile Robotics 2. State Estimation 3. Localization 4. Particle Filters 5 Monte Carlo Simulation

The paper used in this publication may meet the minimum requirements of the American National Standard for Information Sciences — Permanence of Paper for Printed Library Materials, ANSI Z39.48–1984.

10 11

First edition: 30 November 2010

In theory, there is no difference between theory and practice.
In practice, there is.
– Attributed to Yogi Berra and Jan L.A. van de Snepscheut.

Contents

List of Figures

List of Tables

Preface

This tutorial started as a chapter of my Ph.D. thesis [45]. At a next iteration it became available as a techincal report (TR-CIM-04-02) from the Centre for Intelligent Machines, McGill University [46]. It was placed online to help other researchers that are interested in implementing a particle filter for mobile robots. A shorter version of this text was published in the International Conference on Robotics and Automation 2003 (ICRA-2003) [44]. In its current form as a book it is augmented with several different examples that cover different scenarios of mobile robot localization.

The main goal of this book is to introduce the concepts of a Particle Filter while at the same time provide examples and pointers for a practical implementation in the field of robotics. The focus of this book is on discussing the practical aspects of a standard Particle Filter, as such, little to no attention is being paid to the more advance versions of Rao Blackwelized Particle, Filter, FastSLAM 1 and 2, Unscented Particle filter, etc.

Chapter 1

Introduction

The Particle Filters belong in the Sequential Monte Carlo Simulation family of algorithms and have been extensively used for model estimation. They are an instantiation of the Bayesian Filter, therefore, they operate in a recursive function under the assumption of the Markov property.

The following subjects are presented in this tutorial. The next chapter introduces some relevant background on the topics of estimation theory, odometry error modeling, mobile robot localization, and an outline of the Bayesian framework the particle filter is based on. Chapter 3 contains a detailed description of the Monte-Carlo Simulation method (particle filtering) we used in order to implement the Bayesian framework. Chapter 6 presents an odometric error study of a differential drive robot and an analysis of odometric error modeling. Chapter 7 presents different algorithms for the resampling stage of the particle filter.

Chapter 2

Related Work

Particle Filters have been used extensivelly in many different fields. In this chapter relevant background for the easier understanding of the subsequent chapters is reviewed. A brief overview on estimation theory is presented in Section 2.1. Section 2.2 discusses the work on odometric error estimation and dead reckoning, and Section 2.3 presents an overview on localization.

2.1 Estimation Theory

During the exploration of the unknown environment, the robots maintain a set of hypotheses with regard to their position and the position of the different objects around them. The input for updating these beliefs comes from the various sensors the robots poses. An " optimal estimator" [21] can be employed in order for the mobile robots to update their beliefs as accurately as possible. More precisely, the position of an obstacle observed in the past can be updated every time more data become available (a process called smoothing). Moreover, after an action, the estimate of the pose of the robot can be updated based on the data collected up

to that point in time (a process called filtering).

Kalman filtering [8, 21, 41] is a standard approach for reducing the error, in a least squares sense, in measurements from different sources. In particular, in mobile robotics, Smith, Self and Cheeseman provided a framework for estimating the statistical properties of the error in robot positioning given different sets of sensor data [52, 53]. A variation is based on Extended Kalman filtering (EKF), where a nonlinear model of the motion and measurement equations is used [12,34]. Roumeliotis *et al.* successfully employed Extended Kalman Filter in a variety of tasks such as localization and multi-robot mapping [48–50]. Kurazume *et al.* proposed the use of multiple robots, equipped with a sophisticated laser range finder, in order to localize, using some of them as movable landmarks [31–33]. The team of mobile robots uses a swarm behavior, using each other for localization. The fact that two robots could see each other was not used to infer that the space between them was empty.

One approach that has gained popularity lately falls under the category of Monte Carlo Simulation (see Doucet *et al.* [16] for an overview) and is known under different names in different fields. The technique we use was introduced as particle filtering by Gordon *et al.* [23] for tracking a moving target. In mobile robotics particle filtering has been applied successfully by different groups for single robots [13, 14, 28, 56], or for multiple robots [15], during navigation for online localization and for localization with a uniform prior (solving the kidnaped robot problem) [55], but also during exploration and mapping [27]. In vision this technique was introduced under the name of condensation [25] and particle filtering [3] for the estimation of optical flow in image sequences [26] and for tracking multiple moving objects in video sequences [39, 54].

2.2 Dead Reckoning

Dead reckoning is the procedure of modeling the pose (position and heading) of a robot by updating an ongoing pose estimate through some internal measures of velocity, acceleration and time [6,17]. In most mobile robots this is achieved with the use of optical encoders on the wheels and is called odometric estimation. The estimate of the pose of the robot is usually corrupted with errors resulting from conditions such as: unequal wheel diameters, misalignment of wheels, finite encoder resolution (both space and time), wheel-slippage, travel over uneven surfaces [6]. The process of correcting the pose estimate is referred to as *localization*.

Borenstein and Fend in numerous studies present an analysis of the mechanical/kinematic causes of odometry error. Furthermore, they proposes a standard test (*UMB*test) for the estimation of systematic error [7]. Chong and Kleeman [11] use the *UMB*test for the elimination of systematic error and then calculate analytically the Covariance matrix for an extended Kalman Filter. Moon *et al.* [42] studied the effect of speed and acceleration in the kinematics of differential-drive robot, and proposed a method for maintaining a straight line trajectory. Roy et al [51] proposed an online calibration using external sensing in order to estimate the systematic error as a separate component for rotation and for translation.

2.3 Localization

There are two major approaches to localization of a mobile robot based on whether the full structure of the environment is used. For both approaches a variety of sensing methodologies can be used including computational vision, sonar or laser range finding [17].The first approach is to use landmarks in the environment in order to localize frequently and thus reduce the odometry error [6]. A common technique is to select a collection of landmarks in known

positions and inform the robot beforehand [20, 22, 35]. Another technique is to let the robot select its own landmarks according to a set of criteria that optimize its ability to localize, and then use those landmarks to correct its position [2]. The second approach to localization is to perform a matching of the sensor data collected at the current location to an existing model of the environment. Sonar and laser range finder data have been matched to geometrical models [34, 37, 38, 40, 43, 57], and images have been matched to higher order configuration space models [1, 18] in order to extract the position of the robot. Borenstein suggested a two-part robot that would more accurately measure its position by moving one part at a time [4]. Also, Markov models have been used in order to describe the state of the robots during navigation [30].

The existence of clearly identifiable landmarks is an optimistic assumption for an unknown environment. Even in man-made environments, the cost of maintaining labels in prearranged positions may be prohibitive. Moreover, in large-scale explorations the robot may have to travel a large distance (larger than its sensor range) before being able to locate a distinct landmark.

2.4 Bayesian Reasoning

The Bayesian approach provides a general framework for the estimation of the state of our system (the current pose of all robots) in the form of a probability distribution function (*pdf*), based on all the available information.

For the linear-Gaussian estimation problem[1] the required *pdf* remains Gaussian and the Kalman filter provides a provably optimal solution [8, 29, 48]. In the non-linear Gaussian case the Extended Kalman Filter (EKF) has been successfully used by linearizing the control equations [52, 53]. For non-linear, non-Gaussian models

[1] Where the noise probability distribution functions are Gaussian and the model of the system is linear.

two difficulties must be resolved: how to represent a general *pdf* using finite computer storage and how to perform the integrations involved in updating the *pdf* when new data are acquired. During the exploration the uncertainty build-up in the pose estimate of each robot translates into uncertainty in the resulting map. In order to improve the accuracy of the map the pose of the robot has to be estimated at discrete time steps. This is an instance of the discrete time estimation problem and can be formulated in state-space notation (see also Gordon *et al.* [23]).

The i^{th} robot pose at time $t = k$ is represented by the state vector $\mathbf{x}_k^i = [x_k^i, y_k^i, \hat{\theta}_k^i]^T$, $\mathbf{x}_k^i \in \Re^2 \times S^1$. Each robot takes action (α_k^i) and its pose evolves according to Equation 2.1. [2]

$$\mathbf{x}_k = f_\alpha(\mathbf{x}_{k-1}, v_k) \tag{2.1}$$

where, f_α is the system transition function that models how action α probabilistically modifies the pose of the robot and how it is affected by the noise v_k. The actual transfer function f_α is not analytically available; instead, a simulation (as described in chapter 6 Section 6.2) that models the effect of noise and provides an approximation $\hat{f}_\alpha \approx f_\alpha$ is used.

After each action is performed the robot acquires one (or more) sensor readings. Every sensor measurement available at time $t = k$ is included in a sensor data vector noted as $\mathbf{z}_k^i$. These measurements are related to the state vector via the observation equation 2.2.

$$\mathbf{z}_k = g_k(\mathbf{x}_k, u_k) \tag{2.2}$$

where g_k is the measurement function and u_k is the noise model.

[2] The superscript "i" that indicates the robot to which we refer is dropped for clarity of presentation for the rest of the discussion.

It is assumed that the initial *pdf* $P(\mathbf{x}_0)$ is known and that the available information at time $t = k$ is the set of measurements and the set of actions up to that time. In order for the robot to decide the next action it needs to know its current pose or, since knowledge of the true pose is not feasible due to noisy measurements, at least the *pdf* of its pose given the previous actions and observations ($P(\mathbf{x}_k|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k)$). This can be achieved recursively, by first predicting the prior probability of $\mathbf{x}_k$ from the previous pose $\mathbf{x}_{k-1}$ (presuming it is available) and the action taken α_k (see Equation 2.3) and then updating using the latest sensor data $\mathbf{z}_k$ in order to obtain the posterior distribution of the pose $\mathbf{x}_k$ of the moving robot given all available information.

$$P(\mathbf{x}_k|\mathbf{x}_0, \alpha_k, \underbrace{\alpha_j, \mathbf{z}_j}_{j=1\dots k-1}) = \int P(\mathbf{x}_k|\alpha_k, \mathbf{x}_{k-1}) P(\mathbf{x}_{k-1}|\mathbf{x}_0, \underbrace{\alpha_j, \mathbf{z}_j}_{j=1\dots k-1}) d\mathbf{x}_{k-1} \tag{2.3}$$

Note that the $P(\mathbf{x}_k|\alpha_k, \mathbf{x}_{k-1})$ can be derived by the system model (Equation 2.1), the known characteristics of the noise v_{k-1} and the $P(\mathbf{x}_{k-1}|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k-1)$, which is the posterior of $\mathbf{x}$ at time $t = k - 1$.

When new sensory information becomes available we can use Bayes rule in order to update the *pdf* of the moving robot with the latest observations (Equation 2.4). The conditional probability of the sensor measurement $\mathbf{z}_k$ given the pose $\mathbf{x}_k$ from which it was obtained can be estimated by the sensing function g_k and the noise model v_k. Finally the normalizing denominator can be obtained through Equation 2.5.

$$P(\mathbf{x}_k|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k) = \frac{P(\mathbf{z}_k|\mathbf{x}_k) P(\mathbf{x}_k|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k-1)}{P(\mathbf{z}_k|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k-1)} \tag{2.4}$$

$$P(\mathbf{z}_k|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k-1) = \int P(\mathbf{z}_k|\mathbf{x}_k) P(\mathbf{x}_k|\mathbf{x}_0, \alpha_j, \mathbf{z}_j : j = 1 \dots k-1) d\mathbf{x}_k \tag{2.5}$$

Chapter 3

Particle Filter

The main objective of particle filtering is to "track" a variable of interest as it evolves over time, typically with a non-Gaussian and potentially multi-modal *pdf*. The basis of the method is to construct a sample-based representation of the entire *pdf*. A series of actions are taken, each one modifying the state of the variable of interest according to some model. Moreover at certain times an observation arrives that constrains the state of the variable of interest at that time.

Multiple copies (particles) of the variable of interest are used, each one associated with a weight that signifies the quality of that specific particle. An estimate of the variable of interest is obtained by the weighted sum of all the particles. The particle filter algorithm is recursive in nature and operates in two phases: *prediction* and *update*. After each action, each particle is modified according to the existing model (*prediction* stage), including the addition of random noise in order to simulate the effect of noise on the variable of interest. Then, each particle's weight is re-evaluated based on the latest sensory information available (*update* stage). At times the particles with (infinitesimally) small weights are eliminated, a

process called resampling.

More formally, the variable of interest (in our case the pose of the moving robot $\mathbf{x}^k = [x^k, y^k, \hat{\theta}^k]^T$) at time $t = k$ is represented as a set of M samples (the "particles") $(S_i^k = [\mathbf{x}_j^k, w_j^k] : j = 1 \dots M)$, where the index j denotes the particle and not the robot, each particle consisting of a copy of the variable of interest and a weight (w_j^k) that defines the contribution of this particle to the overall estimate of the variable.

If at time $t = k$ we know the *pdf* of the system at the previous instant (time $t = k - 1$) then we model the effect of the action to obtain a prior of the *pdf* at time $t = k$ (prediction). In other words, the *prediction* phase uses a model in order to simulate the effect an action has on the set of particles with the appropriate noise added. The *update* phase uses the information obtained from sensing to update the particle weights in order to accurately describe the moving robot's *pdf*. Algorithm 1 presents a formal description of the particle filter algorithm and the next two subsections discuss the details of prediction and update.

Given a particle distribution, we often need to take actions based on the robot pose. Three different methods of evaluation have been used in order to obtain an estimate of the pose. First, the weighted mean $(P_{est} = \sum_{j=1}^{M} w_j \mathbf{x}_j)$ can be used; second, the best particle (the P_j such that $w_j = max(w_k) : k = 1 \dots M$) and, third, the weighted mean in a small window around the best particle (also called robust mean) can be used. Each method has its advantages and disadvantages: the weighted mean fails when faced with multi-modal distributions, while the best particle introduces a discretization error. The best method is the robust mean but it is also the most computationally expensive.

3.0.1 Resampling

One of the problems that appear with the use of particle filters is the depletion of the population after a few iterations. Most of

Require: A set of Particles for Robot i at time 0: $S_i^0 = [\mathbf{x}_j, w_j : j = 1 \dots M]$.
$W = w_j : j = 1 \dots M$
while (Exploring) **do**
  $k = k + 1$;
  if (ESS$(W) < \beta * M$) **then** {`Particle Population Depleted (Equation 3.2)`}
    Index=Resample(W);
    $S_i^k = S_i^k(Index)$;
  end if
  for ($j = 1$ to M) **do** {`Prediction after action` α}
    $\mathbf{x}_j^{k+1} = \hat{f}(\mathbf{x}_j^k, \alpha)$
  end for
  s=Sense()
  for ($j = 1$ to M) **do** {`Update the weights`}
    $w_j^{k+1} = w_j^k * \mathcal{W}(s, \mathbf{x}_j^{k+1})$
  end for
  for ($j = 1$ to M) **do** {`Normalize the weights`}
    $w_j^{k+1} = \frac{w_j^{k+1}}{\sum_{j=1}^{M} w_j^{k+1}}$
  end for
end while
{`ESS is the Effective Sample Size, see Equation 3.2`}

Algorithm 1: Particle Filter Algorithm; procedures are noted as underlined text, Comments are inside curly brackets "{`comment`}".

the particles have drifted far enough for their weight to become too small to contribute to the *pdf* of the moving robot [1]. If we consider the current set of particles $S_k = \{\mathbf{x}_i^k, w_i^k\} : k = 1 \dots M$ as a discrete representation of the *pdf* of the moving robot-pose, a new

[1] For most practical implementations the weights become zero due to rounding off.

representation $S'_k = \{\mathbf{x'}^k_i, w'^k_i\} : k = 1 \dots M$ is needed such that $\mathbf{x}^k_i = \mathbf{x'}^l_i$ for k, l in $[1, M]$ and weights $(w'^k_i = 1/M)$ that represent the same *pdf*.

Liu *et al.* [36] refer to two different measures that estimate the number of *near-zero-weight* particles: one is the coefficient of variation cv_t^2 (see Equation 3.1) and the second is the effective sample size ESS_t (see Equation 3.2).

$$cv_t^2 = \frac{var(w_t(i))}{E^2(w_t(i))} = \frac{1}{M}\sum_{i=1}^{M}(Mw(i) - 1)^2 \tag{3.1}$$

$$ESS_t = \frac{M}{1 + cv_t^2} \tag{3.2}$$

When the effective sample size drops below a certain threshold, usually a percentage of the number of particles M, then the particle population is resampled, eliminating (probabilistically) the ones with small weights and duplicating the ones with higher weights.

Different methods have been proposed for resampling; three of the most common ones are discussed in chapter 7. In every case the input is an array of the weights of the particles and the output is an array of indices of which particles are going to propagate forward. The requirement is that the *pdf* reconstructed by the resampled population is very close to the one before the resampling. Experimental tests showed no noticeable improvements over the simple select with replacement scheme. In *Select with Replacement* each particle is selected to continue with a probability equal to its weight. We used the approach of Carpenter *et al.* [10] that runs in linear time in the number of particles (see chapter 7 for a description of the algorithm).

3.0.2 Update

After an action (the motion of one of the robots) the robot tracker sensor is employed in order to estimate the pose of the moving robot [2]. The calculations are dependent on the configuration of the robot tracker employed. The next two sections present the update of the weights of the particles of the moving robot for the laser/target robot tracker combination for two different cases. First we derive the update equations when the laser range finder is mounted on the stationary robot (subsection 5.2); second for when the laser range finder is mounted on the moving robot and the target is mounted on the stationary robot (subsection 5.2).

3.1 Dimensionality

Particle Filters are sample based methods, as such their performance depends on the number of samples. The larger the number of samples the more the state space is covered. The number of dimensions of the state space is crucial in the selection of sample size. For example, for covering an one dimensional space, between zero and one, 20 particles provide some good coverage; see 3.1a. While for a two dimensional area, twenty particles appear rather sparse; see 3.1b. Raising the number of particles on the power of the dimensions, $20^2 = 400$ particles for example generate adequate coverage; see 3.1c. when the state space is three dimensional, the trend is similar, twenty particles generate very sparse coverage; see 3.1d, twenty squared, is still sparse; see 3.1e; while twenty cubed ($20^3 = 8000$) particles provide a dense coverage; see 3.1f.

[2] Additional sources of information (e.g. consistency of sensed parts of the environment with the map up to this point) can also be used during the update stage.

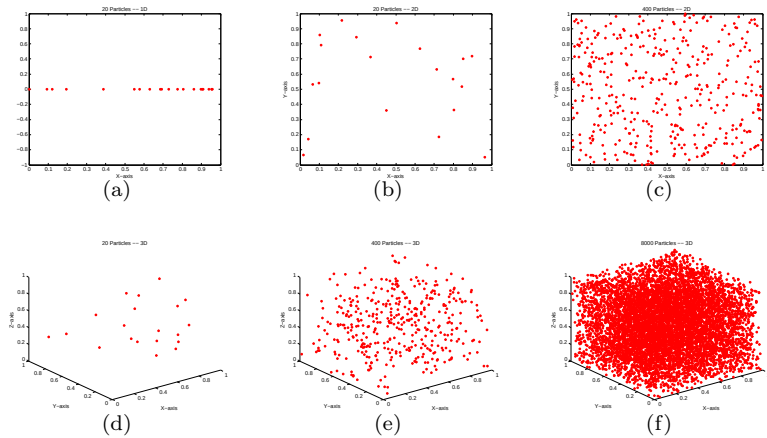


Figure 3.1: Spatial coverage in different dimensions for varying number of particles: (a) 1D, 20 particles; (b) 2D, 20 particles; (c) 2D, 400 particles; (d) 3D, 20 particles; (e) 3D, 400 particles; (f) 3D, 8000 particles.

Chapter 4

Propagation

In this chapter we discuss the propagation step of the algorithm for a variety of examples. Different types of motion schemes are discussed both on 2D and 3D moving robots. The propagation step does not have to have a single equation, if there is a reliable simulator of the robot, this can be used to simulate the propagation step. The very first step is the initialization of the Particle Filter, which is discussed in the next section, then at each propagation step

4.1 Initialization

Depending on the starting assumptions different particle distributions can be used. At the begining each particle is assigned a weight equal to $1/N$ where N is the number of particles. If the robot starts from a known initial pose, the all the particles would be concentrated and have an identical pose; see Fig. 4.1a. It is possible that we know the position with a certain degree of accuracy but not the orientation, this can be easily modelled by drawing the particles from a Normal distribution for the position, with the appropriate

σ, and the orientation uniformally between 0 and 2π; this would be the case if we have information from a GPS device before any motion; see Fig. 4.1b. Finally in the case of global localization, unknown initial pose, then the particles are distributed uniformally over the environment; see Fig. 4.1c. In the examples of Figure 4.1, the known map have been used to reject particles which were created inside obstacles, leaving only valid particles.

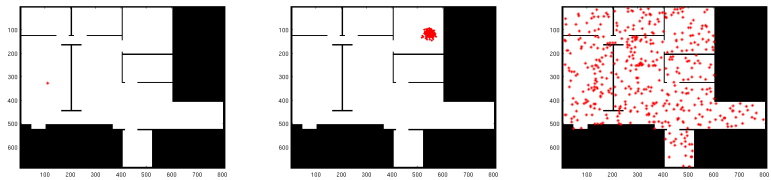


Figure 4.1: Particle distribution for different scenarios: (a) Perfectly known initial pose of the robot. (b) Initial pose known with some uncertainty. (c) Unknown initial pose of the robot.

4.2 Piece-wise linear Motion

4.2.1 Prediction

In order to predict the probability distribution of the pose of the moving robot after a motion we need to have a model of the effect of noise on the resulting pose. Many different approaches have been used (see Borenstein *et al.* [5,7] for an overview), most of which use an additive Gaussian noise model for the motion. Any arbitrary motion $[\Delta x, \Delta y]^T$ can be performed as a rotation followed by a translation (a piecewise linearisation, see Figure 4.2). The robot's initial pose is $[x, y, \hat{\theta}]^T$. First the robot rotates by $\delta\hat{\theta} = \hat{\theta}_k - \hat{\theta}$, where

$\hat{\theta}_k = \arctan(\Delta y/\Delta x)$ to face the destination position, and then it translates forward by distance $\rho = \sqrt{\Delta x^2 + \Delta y^2}$ [1]. If the starting pose is $[x, y, \hat{\theta}]^T$, the resulting pose $[x', y', \hat{\theta}_k]^T$ is given in Equation 4.1. Consequently, the noise model is applied separately to each of the two types of motion because they are assumed independent.

$$\begin{bmatrix} x' \\ y' \\ \hat{\theta}' \end{bmatrix} = \begin{bmatrix} x + \rho\cos(\hat{\theta}_k) \\ y + \rho\sin(\hat{\theta}_k) \\ \hat{\theta}_k \end{bmatrix} \tag{4.1}$$

Rotation

When the robot performs a relative rotation by $\delta\hat{\theta}$ the noise from the odometry error is modeled as a Gaussian with mean (M_{rot}) experimentally established (see appendix 6) and sigma proportional to $\delta\hat{\theta}$. More formally, if at time $t = k$ the robot has an orientation $\hat{\theta}_k$ then after the rotation (time $t = k + 1$) the orientation of the robot is given by Equation 4.2. Therefore, to model the rotation of $\delta\hat{\theta}$, the orientation $\hat{\theta}_j$ of each particle j is updated by adding $\delta\hat{\theta}$ plus a random number drawn from a normal distribution with mean M_{rot} and standard deviation $\sigma_{rot}\delta\hat{\theta}$ ($N(M_{rot}, \sigma_{rot}\delta\hat{\theta})$, where σ_{rot} is in degrees per 360°).

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \delta\hat{\theta} + N(M_{rot}, \sigma_{rot}\delta\hat{\theta}) \tag{4.2}$$

Translation

Modeling the forward translation is more complicated [2]. There are two different sources of error, the first related to the actual

[1] In our experimental setup the Nomadic Technologies Superscout II robots used are controlled by the same rules.

[2] For a detailed description of the model please refer to appendix 6 sections 6.1.2,6.2.2.

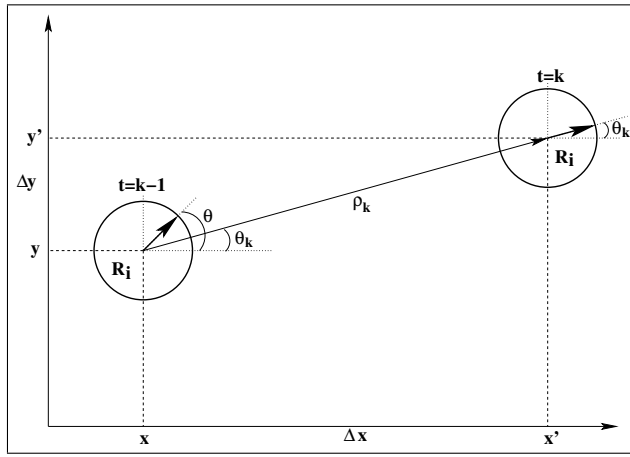
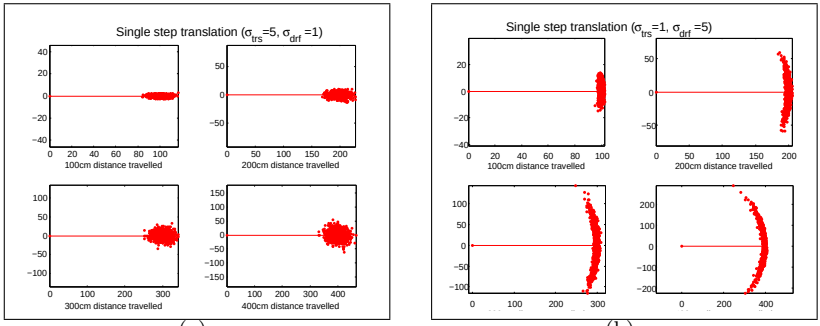


Figure 4.2: Arbitrary motion $[\Delta x, \Delta y]^T$ of robot R_i. At time $t = k-1$ the pose is $[x, y, \hat{\theta}]^T$, after the motion at time $t = k$ the pose is $[x', y', \hat{\theta}_k]^T$. The robot first rotates to orientation $\hat{\theta}_k$ and then translates by ρ_k.

distance traveled and the second related to changes in orientation during the forward translation. During the translation the orientation of the robot changes constantly resulting in a deviation from the desired direction of the translation; such effect is called drift and we model it by adding a small amount of noise to the orientation of the robot before and after each step. As well, if the intended distance is ρ, the actual distance traveled is given by ρ plus some noise following a Gaussian distribution. Experimental results provide the expected value and the standard deviation for the drift and pure translation. Because it is very difficult to analytically model the continuous process, a simulation is used that discretizes the motion to K steps, where K is chosen to be low enough for compu-

tational efficiency but high enough in order to describe the effect of noise in forward translation. If $[\sigma_{translation}, \sigma_{drift}]$ are experimentally obtained values per distance traveled then at each step of the simulation the standard deviation used is given in Equation 4.3. Algorithm 2 provides a formal description of the prediction phase of a set of particles S for a forward translation by distance ρ.

$$\begin{aligned} \sigma_{trs} &= \sigma_{translation}\sqrt{K} \\ \sigma_{drft} &= \sigma_{drift}\sqrt{\frac{K}{2}} \end{aligned} \tag{4.3}$$



(a) (b)

Figure 4.3: The effect of $\sigma_{trs}, \sigma_{drft}$ for the forward translation: (a) $\sigma_{trs} = 5cm/m, \sigma_{drft} = 1^\circ/m$ (b) $\sigma_{trs} = 1cm/m, \sigma_{drft} = 5^\circ/m$.

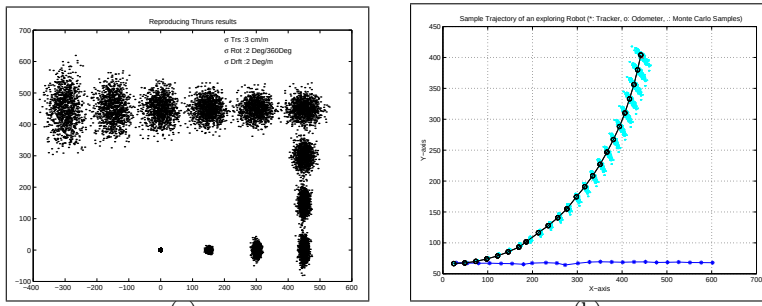
Figure 4.3 presents a graphical illustration of the effect of the two noise parameters $(\sigma_{trs}, \sigma_{drft})$ in the predictive model. In both cases the robot makes a single forward motion of 100cm (upper left sub-plot), 200cm (upper right sub-plot), 300cm (lower left sub-plot), and 400cm (lower right sub-plot). In Figure 4.3a the un-

Input: Set of M Particles::S; Translation distance::ρ
$\delta\rho = \frac{\rho}{K}$;
for ($j = 1$ to M) **do** { `For each particle`}
  for ($k = 1$ to K) **do** { `At each of K steps`}
    E_{trs} =rand_N$(M_{trs} * \delta\rho, \sigma_{trs} * \delta\rho)$;
    E_{drft} =rand_N$(M_{drft} * \delta\rho, \sigma_{drft} * \delta\rho)$;
    $\hat{\theta}[j] = \hat{\theta}[j] + E_{drft}$;
    $x[j] = x[j] + (\delta\rho + E_{trs} * \cos(\hat{\theta}[j])$;
    $y[j] = y[j] + (\delta\rho + E_{trs} * \sin(\hat{\theta}[j])$;
    E_{drft} =rand_N$(M_{drft} * \delta\rho, \sigma_{drft} * \delta\rho)$;
    $\hat{\theta}[j] = \hat{\theta}[j] + E_{drft}$;
  end for
  $S'[j] = [x[j], y[j], \hat{\theta}[j]]^T$;
end for
Return(S')

Algorithm 2: Forward Translation with Noise; rand_N(M, σ) is a pseudo-random number generator drawing samples from a Normal distribution with mean M and standard deviation σ; procedures are noted as underlined text, Comments are inside curly brackets "{comment}". The variables M_{trs} and M_{drft} represent the mean error and are experimentally derived.

certainty in the distance traveled is the dominant uncertainty and thus the particles spread a lot more in the direction of the motion. In contrast, in Figure 4.3b, where the drift noise dominates, the particles spread in a circular pattern. Appendix 6 contains a detailed experimental study of these parameters using the Nomadic Technologies Superscout II mobile platform.

Figure 4.4 presents two examples of complex motions and illustrates the performance of the prediction stage of the particle filter. In figure 4.4a, the robot moves forward three times, rotates ninety degrees, then translates forward three more times, after which it



(a) (b)

Figure 4.4: (a) Large trajectory, the uncertainty build up is represented by the spread of the particle cloud. (b) Series of forward translations and 360° rotations performed in our laboratory. The connected curved line represent the uncorrected odometer values (captured accurately by the cloud of particles), and the bottom line represents the actual trajectory.

rotates again by ninety degrees and translates forward five times. As can be seen the uncertainty grows unbounded. Sub-figure 4.4b presents experimental validation of our predictive model. In this case the predictive model was guided by a set of motion commands that were used in an experiment in our laboratory (for the full description of this experiment please refer to chapter 8 of [45]). In short, the experiment consisted of forward translations, each one followed by four rotations by ninety degrees (in order to sense the environment in four different directions). The connected circles in sub-figure 4.4b represent the uncorrected odometer values. In fact, the actual trajectory of the robot was kept in a straight line but the odometry estimates did deviate due to noise. The predictive model was constructed using the noise statistical parameters col-

lected in our laboratory (see Appendix 6). The predicted cloud of particles can be seen around the recorded values following the trajectory with high accuracy.

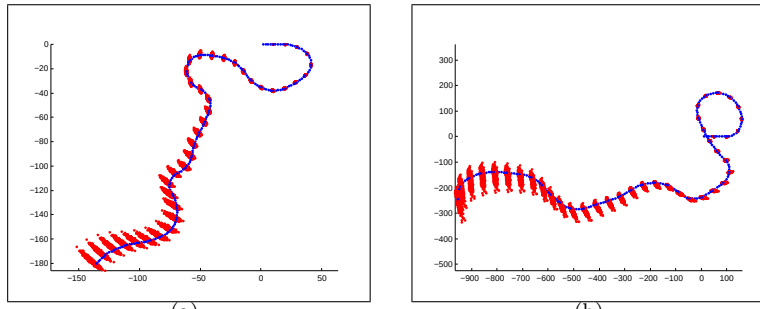
4.3 Linear and Angular Velocity Control

Many mobile robots operate

$$\begin{aligned} x_i^{t+1} &= x_i^t + (v_t + w_{v_t})\delta t \cos(\phi) \\ y_i^{t+1} &= y_i^t + (v_t + w_{v_t})\delta t \sin(\phi) \\ \phi_i^{t+1} &= \phi_i^t + (\omega_t + w_{\omega_t})\delta t \end{aligned} \tag{4.4}$$

where v_t is the linear velocity and ω_t is the angular velocity at time t.

Figure 4.5 presents the spread of particles sampled at 1Hz ($\delta t = 1sec$); plotting one sample every second and plotting all the particles every five seconds. The linear velocity is constant ($v_t = 1m/sec$), and the angular velocity changes randomly every ten seconds.



(a) (b)

Figure 4.5: Sample trajectories using propagation only.

4.4 Failure of Motion Commands

In a planetary exploration scenario, the communication link can be faulty, as such only 60% of the commands arrive. 4.6 presents a simple scenario of piecewise linear motions. The rover starts at [-8, 0, 0°]. Three commands are send to the rover: translate by 4m; see 4.6b, rotate by 30°; see 4.6, finally translate by 6m; see 4.6d. The resulting particle distribution has several modes. Some particles are at [-8, 0, 0°], all three commands failed. Some particles are at [-8, 0, 30°], the two translation commands failed but the rotation command was received. Some particles are at [-4, 0, 0°], the rotation and the second translation failed; while some particles are at [-4, 0, 30°], only the last translation command failed. Some particles are at [-2, 0, 0°], the first translation and the rotation commands failed. Some particles are at $[-8 + 6 * \cos(30°), 6 * \sin(30°), (30°)]$, only the first translation failed. Some particles are at [2, 0, 0°], when only the rotation failed. Finally, when all three commands were successful, the particles arrived at $[-4 + 6 * \cos(30°), 6 * \sin(30°), (30°)]$.

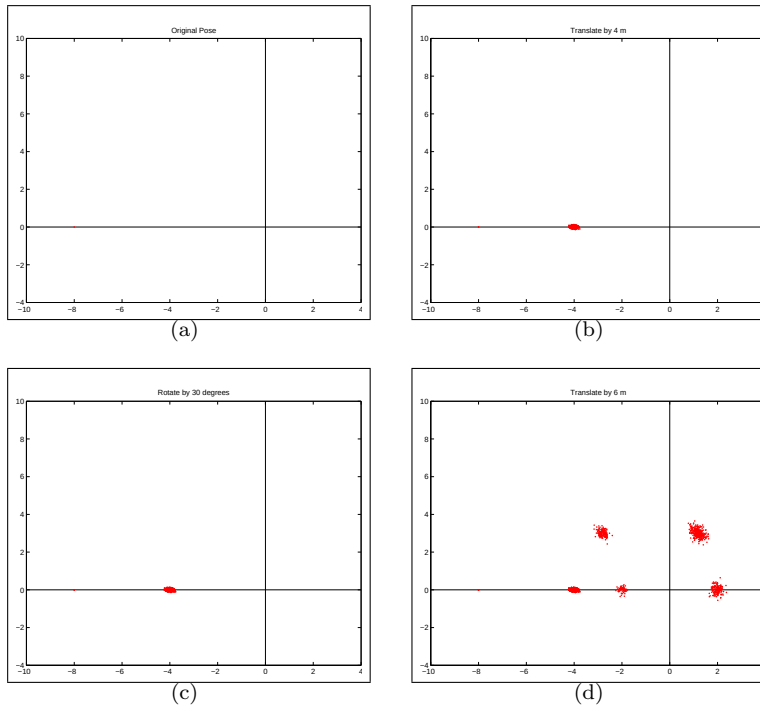


Figure 4.6: (a) Initial pose at [-8, 0, 0°]. (b) Translation by 4m, only 60% of the particles move to [-4, 0, 0], while 40% of the particles stay at [-8, 0, 0°]. (c) Rotate by 30°, 60% of the particles in the two locations rotate by 30°. (d) Translate by 6m, some particles stayed at the original pose, while some stayed at the original position but rotated by 30°, some translate by 6m.

Chapter 5

Update

5.1 Landmark based updated

5.2 Cooperative Localization

Pose Estimation, stationary robot observing moving robot

If the pose of the stationary robot $\mathbf{x}_s = [x_s, y_s, \hat{\theta}_s]^T$ (with laser range finder) and the pose of the moving robot $\mathbf{x}_m = [x_m, y_m, \hat{\theta}_m]^T$ (with target) are known, then the robot tracker sensor measurement $\mathbf{z} = [\rho, \hat{\theta}, \hat{\phi}]^T$ can be calculated by Equation 5.1:

$$\mathbf{z} = \begin{bmatrix} \rho \\ \hat{\theta} \\ \hat{\phi} \end{bmatrix} = \begin{bmatrix} \sqrt{dx^2 + dy^2} \\ atan2(dy/dx) - \hat{\theta}_s \\ atan2(-dy/-dx) - \hat{\theta}_m \end{bmatrix} \tag{5.1}$$

where $dx = x_m - x_s$ and $dy = y_m - y_s$.

If the known information is the pose of the stationary robot ($\mathbf{x}_s$) (with the laser range finder) and the robot tracker measurement is

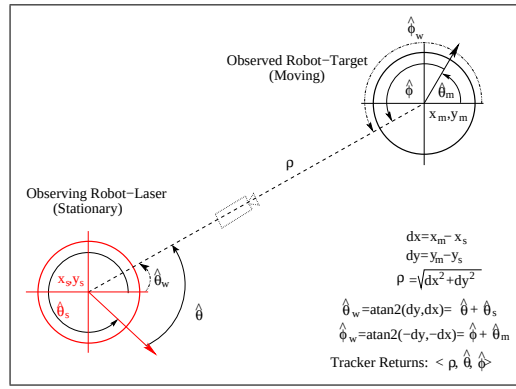


Figure 5.1: The stationary robot with the robot tracker sensor observes the moving robot that carries the target.

($\mathbf{z} = [\rho, \hat{\theta}, \hat{\phi}]^T$) then the *estimate* of the pose of the moving (target) robot ($\mathbf{x}_{m_{est}}(k+1)$) is given in Equation 5.2:

$$\mathbf{x}_{m_{est}}(k+1) = \begin{bmatrix} x_{m_{est}} \\ y_{m_{est}} \\ \hat{\theta_{m_{est}}} \end{bmatrix} = \begin{bmatrix} x_s + \rho\cos(\hat{\theta}_s + \hat{\theta}) \\ y_s + \rho\sin(\hat{\theta}_s + \hat{\theta}) \\ \pi + \hat{\theta} + \hat{\theta}_s - \hat{\phi} \end{bmatrix} \tag{5.2}$$

The Equations 5.1 and 5.2 are equivalent. Consequently, the above equations can be used in order to calculate the weight of each particle of the moving robot, assuming a Gaussian error model for each component of the sensor data $(\rho, \hat{\theta}, \hat{\phi})$, in two different ways. First, let the i^{th} particle at time k be $\mathbf{x}^k_{m_i} = [x_{m_i}, y_{m_i}, \hat{\theta}_{m_i}]^T$. Then if the pose of the stationary robot is known $\mathbf{x}_s = [x_s, y_s, \hat{\theta}_s]^T$ the estimated tracker measurement $\mathbf{z}_i$ for particle i is given in Equation 5.3:

$$\mathbf{z}_i = \begin{bmatrix} \rho_i \\ \hat{\theta}_i \\ \hat{\phi}_i \end{bmatrix} = \begin{bmatrix} \sqrt{dx_i^2 + dy_i^2} \\ atan2(dy_i, dx_i) - \hat{\theta}_s \\ atan2(-dy_i, -dx_i) - \hat{\theta}_{m_i} \end{bmatrix} \tag{5.3}$$

where $dx_i = x_{m_i} - x_s$ and $dy_i = y_{m_i} - y_s$.

The weight for particle i then is proportional to the probability of $\mathbf{x}_{m_i}^{k+1}$ given $\mathbf{x}_s$ and $\mathbf{z}_i$ (see Equation 5.4). As can be seen in Equation 5.3 the value of $\hat{\phi}_i$ is affected by the complete pose of particle i (both position and orientation). Therefore the error in position (x_{m_i}, y_{m_i}) is used twice. In Equation 5.4 the constants $\sigma_\rho, \sigma_{\hat{\theta}}, \sigma_{\hat{\phi}}$ are the presumed standard deviation of the robot trackers measurement noise and they signify the confidence with which we weight each measurement.

$$P(\mathbf{x}_{m_i}^{k+1} | \mathbf{x}_s, \mathbf{z}) = \frac{1}{\sqrt{2\pi}\sigma_\rho} e^{\frac{-(\rho-\rho_i)^2}{2\sigma_\rho^2}} \frac{1}{\sqrt{2\pi}\sigma_{\hat{\theta}}} e^{\frac{-(\hat{\theta}-\hat{\theta}_i)^2}{2\sigma_{\hat{\theta}}^2}} \frac{1}{\sqrt{2\pi}\sigma_{\hat{\phi}}} e^{\frac{-(\hat{\phi}-\hat{\phi}_i)^2}{2\sigma_{\hat{\phi}}^2}} \tag{5.4}$$

Figure 5.2 illustrates the spatial variation of Equation 5.4. In particular the spatial variation of the contribution of each component $(\rho, \hat{\theta}, \hat{\phi})$ to the weighting function is presented in the first three sub-plots, and the spatial variation of the weighting function is presented on the lower right sub-plot. For clarity of presentation, the pose of the observing robot is set at $\mathbf{x}_s = [0, 0, 0]^T$ and the pose of the moving robot at $\mathbf{x}_m = [100, 100, 45]^T$, and using Equation 5.1 the tracker measurement $\mathbf{z} = [\rho, \hat{\theta}, \hat{\phi}]^T$ is calculated. Then the spatial variation of the different terms of the product in Equation 5.4 is plotted keeping the moving robots orientation at the correct value (45°).

Experimental results have shown that the accuracy of the position of the robot is (*almost*) fixed (independent of the distance

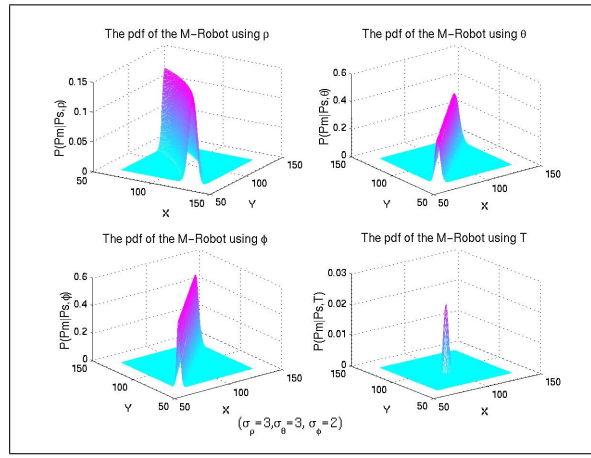


Figure 5.2: The contribution of each measurement of the robot tracker in the weighting *pdf* of the moving robot.

at which the observed robot is seen). Unfortunately, the tracker measurements are in polar coordinates and thus for a fixed error in the angle ($\hat{\theta}$) the longer the distance (ρ) the higher the error. In practice, it is necessary to calculate $\sigma_{\hat{\theta}}$ as a function of ρ:

$$\sigma_{\hat{\theta}} = h(\rho, \sigma_{\hat{\theta}}) = asin(\sigma_{\hat{\theta}}/\rho) \tag{5.5}$$

If the $\sigma_{\hat{\theta}}$ is kept at a fixed value then the weighting function is spread out, as can be seen in the upper right sub-plot in Figure 5.3, and the prediction is less accurate. Figure 5.3 presents the spatial variation of the weighting function for the same condition as in Figure 5.3, except $\sigma_{\hat{\theta}}$ is not scaled.

An alternative weighting function is to use the difference in Cartesian coordinates and the orientation estimate in order to

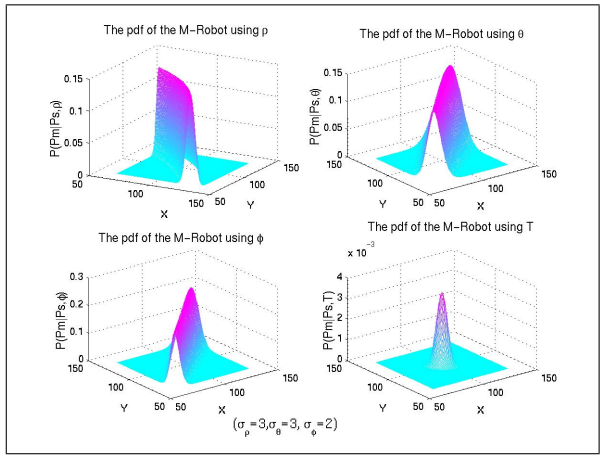


Figure 5.3: The contribution of each measurement of the robot tracker in the weighting *pdf* of the moving robot. In contrast to Figure 5.2 the $\sigma_{\hat{\theta}}$ is not calculated to be proportional to distance between the two robots.

weight the particle $\mathbf{x}_{m_i}^k$ given $\mathbf{x}_s$ and $\mathbf{z}_i$ (see Equation 5.6).

$$P(\mathbf{x}_{m_i}^{k+1}|\mathbf{x}_s,\mathbf{z}) = \frac{1}{\sqrt{2\pi}\sigma_\rho} e^{\frac{-(dx-dx_i)^2}{2\sigma_\rho^2}} \frac{1}{\sqrt{2\pi}\sigma_\rho} e^{\frac{-(dy-dy_i)^2}{2\sigma_\rho^2}} \frac{1}{\sqrt{2\pi}\sigma_{\hat{\phi}}} e^{\frac{-(\hat{\theta}_m-\hat{\theta}_{m_i})^2}{2\sigma_{\hat{\theta}'}^2}} \tag{5.6}$$

The second approach is to use Equation 5.2 and weight every particle depending on how far it is from the estimated pose of the moving robot (see Equation 5.7). Where if $\mathbf{x}_{m_{est}}(k+1) = [x_{m_{est}}, y_{m_{est}}, \hat{\theta}_{m_{est}}]$ is the estimate pose and $\mathbf{x}_{m_i}^k = [x_{m_i}, y_{m_i}, \hat{\theta}_{m_i}]^T$ is the "i^{th}" particle then $d_i = \sqrt{(x_{m_{est}} - x_{m_i})^2 + (y_{m_{est}} - y_{m_i})^2}$. The disadvantage of this approach is that $\sigma_d, \sigma_{\hat{\theta}}$ do not represent

the sensor's noise model.

$$P(\mathbf{x}_{m_i}^{k+1}|\mathbf{x}_s,\mathbf{z}) = \frac{1}{\sqrt{2\pi}\sigma_d} e^{\frac{-(d_i)^2}{2\sigma_d^2}} \frac{1}{\sqrt{2\pi}\sigma_{\hat{\theta}}} e^{\frac{-(\hat{\theta}_{m_{est}} - \hat{\theta}_{m_i})^2}{2\sigma_{\hat{\theta}}^2}} \tag{5.7}$$

During the estimation of the weight the pose of the stationary robot $\mathbf{x}_s$ is used. As the actual pose is not known, different estimates $\bar{\mathbf{x}}_s$ can be employed. The following options have been considered:

- The best particle (the one with maximum weight):

  $$\bar{\mathbf{x}}_s = \mathbf{x}_s^{max}$$

- Weighted Mean:

  $$\bar{\mathbf{x}}_s = \sum_{j=1}^{M} \mathbf{x}_j^s w_j$$

- Use every particle of the stationary robot ($O(n^2)$):

  $$P(\mathbf{x}_{m_i}^{k+1}|\mathbf{x}_s,\mathbf{z}) = \Sigma_{j=1}^{n} P(\mathbf{x}_{m_i}^{k+1}|\mathbf{x}_s^j,\mathbf{z})$$

- Robust Mean: Select only the particles that are less than ϵ from the particle with maximum weight. The advantage of this method is that it selects the mode of the distribution and reduces the discretization error (which occurs when only a single particle is used).

  $$\bar{\mathbf{x}}_s = \sum_{j=1}^{K} \mathbf{x}_s^j w_j : |\mathbf{x}_s^j - \mathbf{x}_s^{max}| \leq \epsilon$$

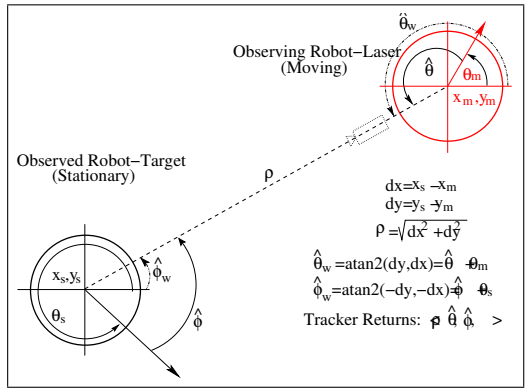


Figure 5.4: Observation.

Pose Estimation, moving robot observing stationary robot

This time the stationary robot has the target. If the poses of the two robots ($\mathbf{x}_s = [x_s, y_s, \hat{\theta}_s]^T$ and $\mathbf{x}_m = [x_m, y_m, \hat{\theta}_m]^T$) are known then the robot tracker sensor measurement ($\mathbf{z} = [\rho, \hat{\theta}, \hat{\phi}]^T$) can be calculated by Equation 5.8 (exactly as in the previous case Equation 5.1).

$$\begin{bmatrix} \rho \\ \hat{\theta} \\ \hat{\phi} \end{bmatrix} = \begin{bmatrix} \sqrt{dx^2 + dy^2} \\ atan2(dy, dx) - \hat{\theta}_m \\ atan2(-dy, -dx) - \hat{\theta}_s \end{bmatrix} \tag{5.8}$$

where $dx = x_s - x_m$ and $dy = y_s - y_m$ [1].

If the pose of the stationary robot ($\mathbf{x}_s$) (carrying the target) and the robot tracker measurement ($\mathbf{z} = [\rho, \hat{\theta}, \hat{\phi}]^T$) are known then

[1] Note that dx, dy are different from Equation 5.1.

the *estimate* of the pose of the moving (carrying the laser) robot ($\mathbf{x}_m$) is given in Equation 5.9.

$$\mathbf{x}_{m_{est}}(k+1) = \begin{bmatrix} x_{m_{est}} \\ y_{m_{est}} \\ \hat{\theta}_{m_{est}} \end{bmatrix} = \begin{bmatrix} x_s + \rho * \cos(\hat{\phi} + \hat{\theta}_s) \\ y_s + \rho * \sin(\hat{\phi} + \hat{\theta}_s) \\ \pi + \hat{\phi} + \hat{\theta}_s - \hat{\theta} \end{bmatrix} \tag{5.9}$$

Applying the same methodology as in the previous section the weight update functions are identical with the ones in Equations 5.4, 5.6, 5.7.

5.2.1 Effect of different Robot Tracker Sensors

Several simple sensing configurations for a robot tracker are available. For example, simple schemes using a camera allow one robot to observe the other and provide different kinds of positional constraints such as the distance between two robots and the relative orientations. In this section we consider the effect the group size has on the accuracy of the localization for different classes of sensors. The experimental arrangement of the robots is simulated and is consistent across all the sensing configurations. The robots start in a single line and they move abreast one at a time, first in ascending order and then in descending order for a set number of exchanges. The selected robot moves for 5 steps and after each step cooperative localization is employed and the pose of the moving robot is estimated. Each step is a forward translation by 100cm. Figure 5.5 presents a group of three robots, after the first robot has finished the five steps and the second robot performs the fifth step.

5.2.2 Range Only

One simple sensing method is to return the relative distance between the robots. Such a method has been employed by [24] in the

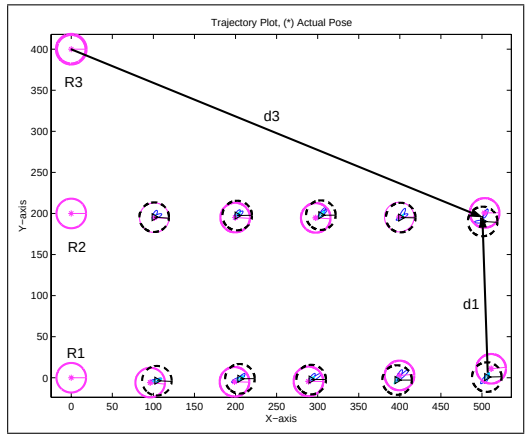


Figure 5.5: Estimation of the pose of robot R2 using only the distance from robot R1 (d1) and from robot R3 (d3).

millibots project where an ultra-sound wave was used in order to recover the relative distance. In order to recover the position of one moving robot in the frame of reference of another, at least two stationary robots (that are not collinear with the moving one) are needed thus the minimum size of the group using this scheme is three robots.

The distance between two robots can be easily and robustly estimated. In experimental simulations, the distance between every pair of robots was estimated and Gaussian, zero mean, noise was added with $\sigma_\rho = 2cm$ regardless the distance between the two robots. Figure 5.6 presents the mean error per unit distance traveled for all robots, averaged over 20 trials. As can be seen in Figure 5.6 with five robots, the positional accuracy is acceptable with an error of 20cm after 40m traveled; for ten robots the accuracy of the localization is very good.

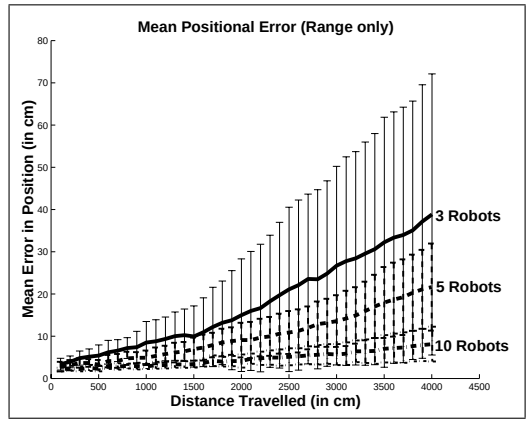


Figure 5.6: Average error in position estimation using the distance between the robots only (3,4 and 10 robots; bars indicate standard deviation).

5.2.3 Azimuth (Angle) Only

Several robotic systems employ an omnidirectional vision sensor that reports the angle at which another robot is seen. This is also consistent with information available from several types of observing systems based on pan-tilt units. In such cases the orientation at which the moving robot is seen can be recovered with high accuracy. We performed a series of trials using only the angle at which one robot is observed, with groups of robots of different sizes. As can be seen in Figure 5.7 the accuracy of the localization does not improve as the group size increases. This is not surprising because small errors in the estimated orientation of the stationary robots scale non-linearly with the distance. Thus after a few exchanges the error in the pose estimation is dominated by the error in the orientation of the stationary robots.

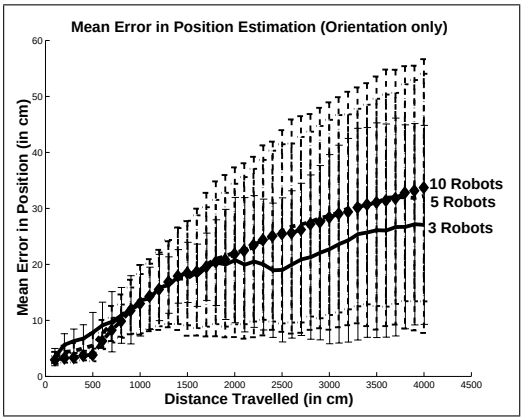
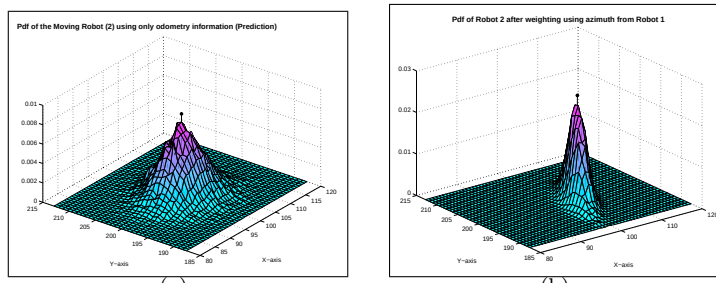


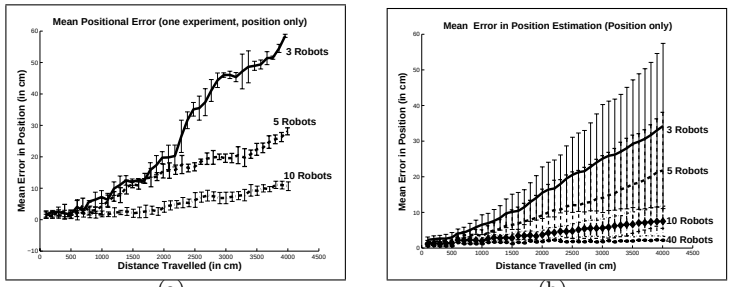
Figure 5.7: Average error in position estimation using the orientation of the moving robot is seen by the stationary ones.

To illustrate the implementation of the particle filter, we present here the probability distribution function (*pdf*) of the pose of the moving robot after one step (see Figure 5.8). The robot group size is three and it is the middle robot R2 that moves. The predicted *pdf* after a forward step can be seen in the first sub-figure (5.8a) using odometry information only; the next two sub-figures (5.8b,5.8c) present the *pdf* updated using the orientation at which the moving robot is seen by a stationary one (first by robot R1 then by robot R3); finally, the sub-figure 5.8d presents the final *pdf* which combines the information from odometry and the observations from the two stationary robots. Clearly the uncertainty of the robot's position is reduced with additional observations.



(a) (b) (c) (d)

Figure 5.8: The *pdf* of the moving robot (R2) at different phases of its estimation: (a) prediction using odometry only; (b) using the orientation from stationary robot R1; (c) using the orientation from stationary robot R3; (d) final *pdf*.



(a) (b)

Figure 5.9: Average error in position estimation using both the distance between the robots and the orientation the moving robot is seen by the stationary ones. (a) Average error in positioning of the team of robots one trial (3,5 and 10 robots). (b) Average error in position estimation over twenty trials (3,5,10 and 40 robots).

5.2.4 Position Only

Another common approach is to use the position of one robot computed in the frame of reference of another (relative position). This scheme has been employed with two robots (see [9]) in order to reduce the uncertainty. The range and azimuth information ($[\rho, \theta]$) is combined in order to improve the pose estimation. As can be seen in Figure 5.9a even with three robots the error in pose estimation is relatively small (average error 30cm for 40m distance traveled per robot, or 0.75%). In our experiments the distance between the two robots was estimated and, as above, zero-mean Gaussian noise was added both to distance and to orientation with $\sigma_\rho = 2cm$ and $\sigma_\theta = 0.5^\circ$ respectively. The experiment was repeated twenty times and the average error in position is shown in Figure 5.9b for groups of robots of size 3,5,10 and 40.

5.2.5 Full Pose

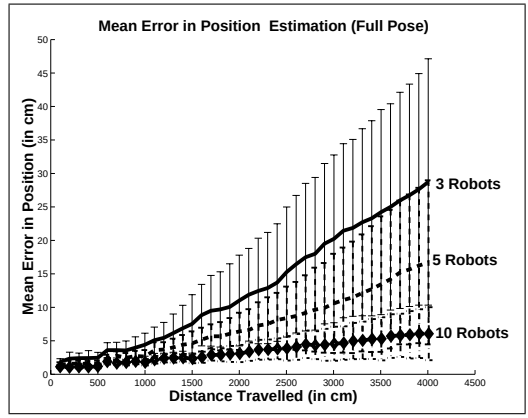


Figure 5.10: Average error in position estimation using full pose $[\rho, \theta, \phi]$.

Some robot tracker sensors provide accurate information for all three parameters $[\rho, \theta, \phi]$ and they can be used to accurately estimate the full pose of the moving robots (see [31, 47]). In the experimental setup the robot tracker sensor was characterized by Gaussian, zero mean, noise with $\sigma = [2cm, 0.5°, 1°]$. By using the full Equation 5.4 we weighted the *pdf* of the pose of the moving robot and performed a series of experiments for 3, 5 and 10 robots. As can be seen in Figure 5.10 the positional accuracy is consistently lower than in the case of range only, orientation only and position only measurements.

In addition, experiments were conducted for larger group sizes and for longer distances traveled. Figure 5.11 presents the mean error over thirty experiments for 3,5,10,15,20 and 30 robots. The mean positional error was calculated as a function of the group

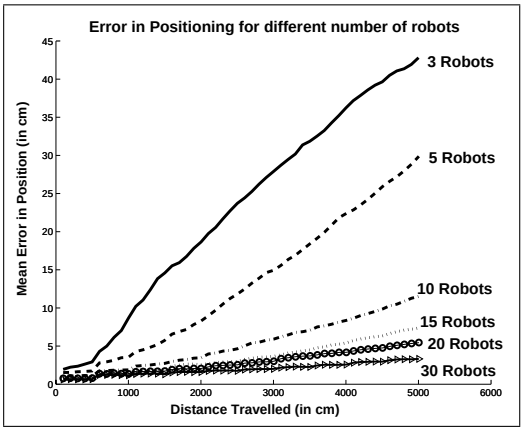


Figure 5.11: Average error in position estimation using full pose $[\rho, \theta, \phi]$ for different number of robots.

size in order to examine the contribution of each additional robot to localization. Two different functions were used in order to model the error with respect to the group size (N) (a) $\mathcal{E}_a(N) = \alpha N^{\beta} + \gamma$ and (b) $\mathcal{E}_b(N) = \alpha e^{\beta N} + \gamma$. Using cross-validation [2] $\mathcal{E}_a(N)$ was selected because it had smaller mean squared error. For a fixed distance traveled (50m) the error function is given in Equation 5.10. As expected the incremental benefit of each additional robot is a function decreasing asymptotically to zero.

$$\mathcal{E}_a(N) = 126.866 N^{-0.948} \tag{5.10}$$

[2]The two functions were fitted for robot group sizes of 3-10,15,20 and 30 (11 group sizes in total), each time omitting one group size and then calculating the difference between the observed error value and the function response.

5.2.6 Example

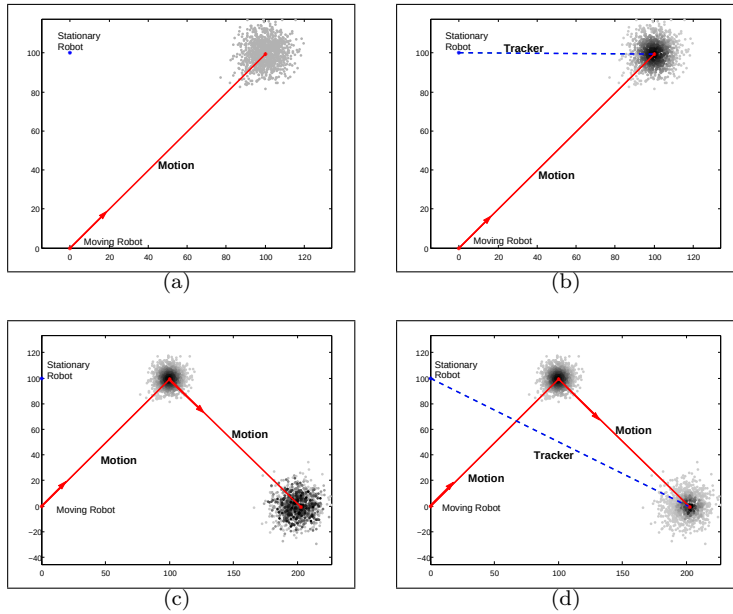


Figure 5.12: (a) *Prediction* of the first step. (b) *Update* using the robot tracker. (c) *Prediction* of the second step. (d) *Update* using the robot tracker.

Figure 5.12 presents an illustration of the above described process over two iterations. The first column present the *prediction* phase and the second column the *update* phase. The moving robot starts at position [0,0], and the stationary robot is located at [0,100]. At figure 5.12a the moving robot moves by [100cm,100cm] and the particles form a cloud of approximately 20cm in radius.

Figure 5.12b presents the update phase based on the tracker sensor measurement (darker color represents higher weights). At the second step the robot moves by [100cm,-100cm] and figure 5.12c presents the cloud of particles. It is worth noting that the particles with higher weights (darker grey) have spread out. Finally, figure 5.12d presents the second update phase where again the particles closer to the sensed pose have higher weights.

Chapter 6

Odometry Error Study

In this chapter we consider the measurement of odometric uncertainty for a mobile robot. The primary emphasis is to experimentally estimate the rate of odometry error buildup in a small differential-drive research robot, and to model its behavior probabilistically. Although the use of Kalman filters and related techniques are common place for robotic systems, it is not uncommon for mobile robotics practitioners to merely make educated guesses not only for the rate of error accumulation for their robots, but also for the error model itself. While there are a few notable papers that rigorously consider error measurement for mobile robots [5, 7, 42], the most common error model used in practice is an unrealistic univariate two-dimensional Gaussian. Furthermore, in simulated environments very crude odometry error models are used, if the error is modeled at all.

Our goal was to develop a more realistic odometry error model that would reflect (at least partially) the complexity of the robot's locomotion. Such a model is used to describe faithfully the probability distribution function of the robot's pose after an arbitrary motion. The odometry error study presented here in combination

with the proposed model provides a practical framework for the implementation of realistic odometry error in different simulation packages. Our primary experimental data is obtained from a differential-drive robot, although we believe the proposed probabilistic model applies to other types of drive mechanism and we have tested it informally on synchro-drive systems as well. The odometry error is detected using a calibrated laser range finder.

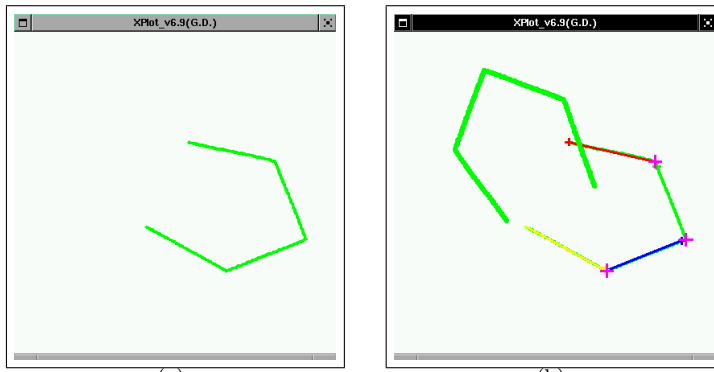
Figure 6.1: Measuring the odometry error on carpet.

6.1 Odometry Study of a Differential Drive Robot

As a baseline we consider the odometry error accrued under various conditions by a commercial differential-drive robot, the Nomadic Technologies Superscout II. This robot uses two wheels to provide

differential drive and odometry feedback with a third rear-mounted castor wheel for balance. Without loss of generality any arbitrary motion by $\Delta X, \Delta Y$ can be achieved by combining a rotation that points the robot towards the target location, followed by a translation that moves the robot to the target location. Therefore we divided the observations into rotational errors and translational errors.

6.1.1 Rotation



(a) (b)

Figure 6.2: The four walls providing three landmarks. (a) Before the rotation. (b) After the rotation.

Empirical knowledge suggests that the largest factor in odometry error is the rotational error[1]. In order to measure the rotational error we placed the robot inside a "C"-shaped enclosure consisting

[1]While we make this observation empirically, it follows naturally from the kinematics of the robot and a simple model for uncertainty in wheel velocity.

of four walls (see Figure 6.1,6.2a). The intersections of the four walls provide three geometric landmarks detectable both in world coordinates and "raw" laser coordinates (see Chapter 6 section 3.2 of [45]). Moreover, the orientations of the four walls in world coordinates should change by the amount of the robot's rotation. To estimate the error, the three landmarks are detected then the robot rotates and the three landmarks are detected again (see Figure 6.2b). The three landmarks in laser coordinates provide three estimates for the rotation and the orientations of the four walls provide four more estimates. The seven estimates are kept only if they all agree up to 0.2 degree. We proceed to measure the rotational error for different motion parameters (rotation angle, speed, acceleration) and on different surfaces.

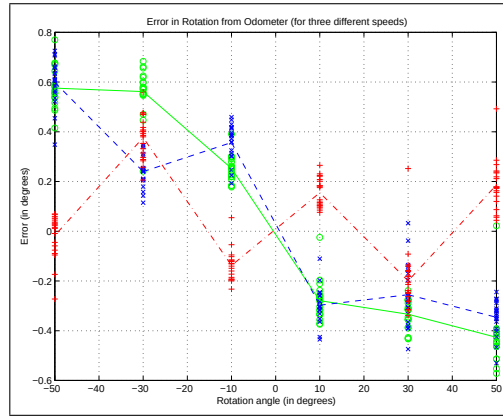


Figure 6.3: Error in rotation relative to the odometer for different angles and for different speeds ("o" speed 10, "x" speed 50, "+" speed 90, lines connect the mean values).

First, we measured the error in rotation for different rotation

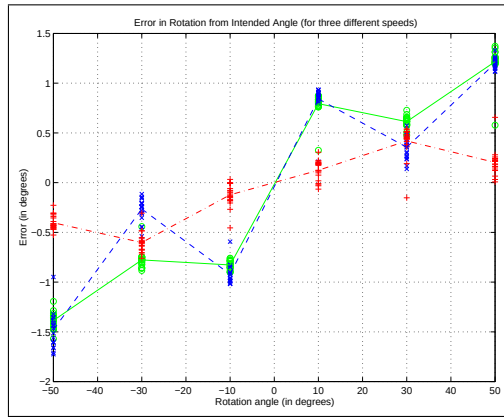


Figure 6.4: Error in rotation relative to the intended pose for different angles and for different speeds (as in Figure 6.3)

and translation speeds and for different angles. Figures 6.3,6.4 present the error measurements relative to the odometer estimate (Figure 6.3) and relative to the intended pose (Figure 6.4); for every speed/angle we gather twenty samples. It is worth noting that they are concentrated (small standard deviation) around a non zero mean value. Moreover from Figures 6.3,6.4 it is clear that a systematic error occurs that biases the error by the direction of the rotation (negative rotation have positive mean error). As it was expected the small rotations provide negligible error. Surprisingly though, the higher speed produced less odometry error ("+" in the figures).

The effect of different surfaces on the rotational error was studied next. Four different surfaces were tested for a rotation of $-90°$ and forty samples were collected each time. The two types of carpets follow more closely a normal distribution than the other two

surfaces. This is due to the fact that the surface is smooth contrary to the tile floor that contains bumps. The friction between the wheels of the robot and the floor (or the carpet) was relatively similar. On the contrary the plastic surface provided less friction thus significantly increasing the rotational error.

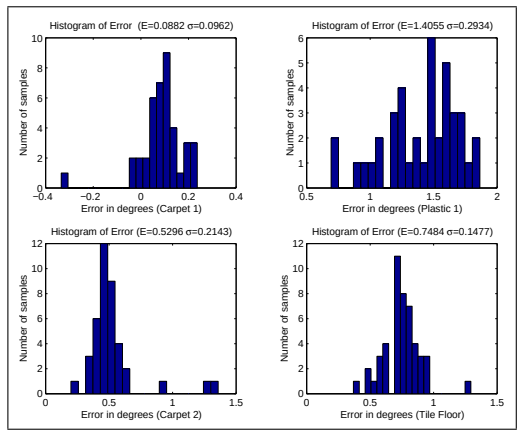


Figure 6.5: Error distribution from the odometry measurement for different surfaces (rotation of 90°).

From Figure 6.6 we see that the error from the intended rotation is much larger. Even though the odometer reported a pose different than the intended one, the control software of the robot stopped the rotation. For applications that require precise positioning, this extra error should be taken into account.

From the results described above we can deduce that a study of the odometry error of the particular mobile robot used is essential in order to model the systematic error that occurs during rotations. A zero mean Gaussian representation would require an unnecessarily large standard deviation forcing us to consider poses

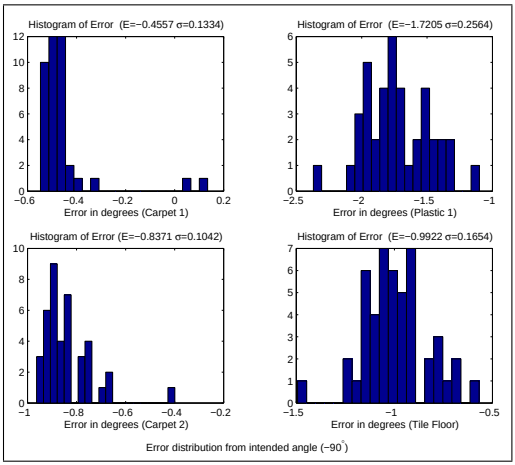


Figure 6.6: Error distribution from the intended pose for different surfaces (rotation of 90°).

of the robot that are in fact unlikely.

6.1.2 Translation

The same setup used for the estimation of the rotational error is used also for the translation. The same enclosure was used (see Figure 6.1). The robot was moved forward by a distance D over different surfaces and with different speeds. After every translation the robot was translated back (by -D) and the pose of the robot was reset to the origin ($P_r = [x_r, y_r, \theta_r]^T = [0, 0, 0]^T$). Figure 6.7 presents the error accumulated after an intended translation of 100cm. The robot was moved 165 times over different areas of our lab (tiled floor). The first three sub-plots present a histogram of the error along the X and Y axis and for the orientation Θ. The

fourth sub-plot present the spatial distribution of the robot poses for all the motions.

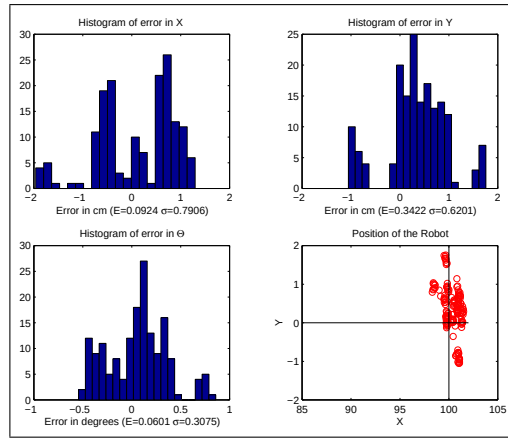


Figure 6.7: Error distribution after translation of 100cm. Tile floor, 165 samples.

Speed	20		60		100	
	M	σ	**M**	σ	**M**	σ
X	-1.843	0.372	-1.850	0.363	-2.266	0.526
Y	-0.863	0.317	-0.977	0.491	-1.041	0.491
Θ	0.587	0.215	0.760	0.366	0.107	0.314

Table 6.1: Mean error and Standard Deviation along the X,Y-axis (in cm) and orientation Θ (in degrees) after the translation of 100cm for three different speeds.

Table 6.1 illustrates the effect of speed in the accumulation of

odometry error for three different speeds (20, 60, 100) during the translation of 100cm along the x-axis. There is a significant increase when the higher speed was used, especially in the systematic error as it manifests in the mean error along the axis of translation. The observations are consistent with the work presented by Moon *et al.* [42] where higher acceleration gives reduced orientation error. As can be seen in Table 6.1 the high acceleration results in small orientation error but higher error along the direction of the translation.

The measurement of odometry error over different surfaces is presented next. Figure 6.8 presents the results for a translation of 120cm on a plastic surface. The same behavior as with the rotation manifests during the translation, with the error in the distance traveled (X-axis) much higher than the error on carpet or tile floor. Figure 6.9 presents the results for the translation on carpet.

The statistical properties of the odometry error collected above enable us to create a realistic error model for the type of robot used. Furthermore, the odometry error measurements could be utilized in the construction of realistic simulation experiments.

6.2 Odometry Error Modeling

In the past little attention has been paid to the modeling of odometry error. The computing power was not enough to permit a precise modeling forcing early researchers to a simple Gaussian *pdf* around the final position of the moving robot as the most general error model. With the computing power currently available, even on board autonomous robots, more elaborate techniques such as condensation (a Monte-Carlo simulation method) and multiple Gaussians are used in order to track the accumulation of uncertainty during motion. In many cases, however, the error model is still based on a single random variable drawn from a normal

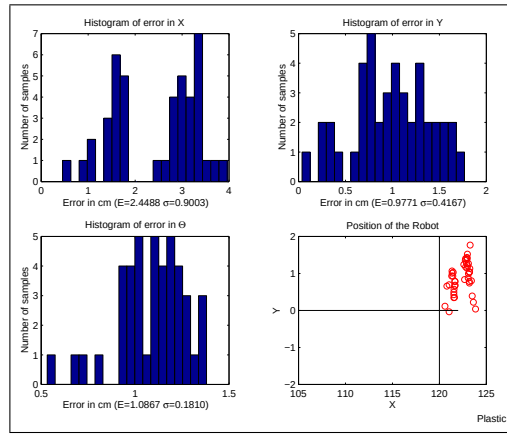


Figure 6.8: Error distribution after translation of 120cm. Plastic surface, 43 samples.

distribution.

There are many sources of error that contribute to the accumulation of uncertainty during motion such as wheel slippage, difference in the diameters of the wheels and anomalies of the floor [2]. Without loss of generality any arbitrary motion by $\Delta X, \Delta Y$ can be achieved by combining a rotation that points the robot towards the target location, followed by a translation that moves the robot to the target location.

For modeling purposes the odometry error could be divided into rotational error [3] and translational error. These errors can be modeled statistically by random variables drawn from three Gaussians with zero mean and $\sigma_{rot}, \sigma_{trans}, \sigma_{drift}$ standard deviations. The

[2] For a more detailed study please refer to Borenstein [5, 19].

[3] For simplicity's sake it is assumed that only the orientation of the moving robot is affected.

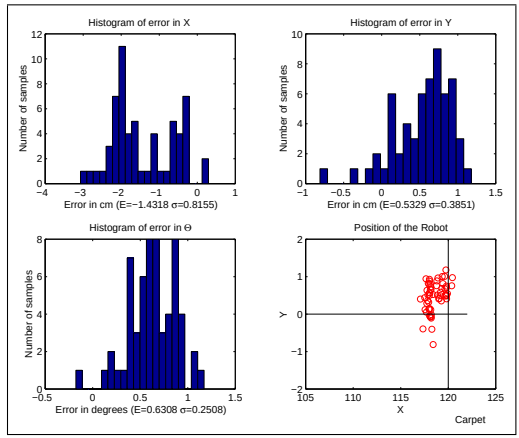


Figure 6.9: Error distribution after translation of 120cm. Carpet surface, 43 samples.

first Gaussian models the error accumulated during pure rotations of the robot. The other two Gaussians model the error that occurs during a forward translation of the robot and affects the complete pose of the moving robot. It is worth noting that an additional source of error could be added that would represent bumps on the floor and small collisions by adding some "salt and pepper" noise.

6.2.1 Rotation

As we saw in section 4.2.1 the noise model for rotational is straight forward described by the general equation 6.1.

$$\theta_{k+1} = \theta_k + \Delta\theta + N(M_{rot}, \sigma_{rot}\frac{\Delta\theta}{360}) \tag{6.1}$$

6.2.2 Translation

Modeling the translation of the robot is more difficult because the noise model is more complex. During a translation by a distance R towards the orientation the robot two kinds of uncertainty accumulate: first, the distance the robot traveled is given by R plus an error. Second, the orientation of the robot constantly changes adding the equivalent of a Brownian type of noise to the final position. While for the real robot the drift is a continuous process that affects the complete trajectory of the translation during simulations, but more important during the modeling of uncertainty, a discretization of the process is required. The simplest approximation [4] of the above process is to model the translation as a partial rotation followed by a translation followed by a second rotation (Fig. 6.10). The reason for this is that the robot would deviate from the trajectory, hence the initial rotation by a small angle, and also the final orientation of the robot is corrupted by some noise, hence the second rotation.

Orientation: For a single translation modeled as one step the orientation of the robot at the beginning would be θ_i and at the end the orientation is $\theta_{i+1} = \theta_i + \mathcal{E}_{\theta_1} + \mathcal{E}_{\theta_2}$, where $\mathcal{E}_{\theta_1}$ and $\mathcal{E}_{\theta_2}$ are the amount of the two rotations that occur before and after the translation (see Fig. 6.10). From experimental data we could have an estimate about the standard deviation of the orientation as a function of the distance traveled (σ_{drift} in degrees per meter traveled). The standard deviation of the orientation after one translation can be calculated in terms of the characteristics of the noise $\mathcal{E}_{\theta_j}, j = 1, 2$, and if $\mathcal{E}_{\theta_1} = \mathcal{E}_{\theta_2} = \mathcal{E}_{\theta_j}$ then the standard deviation of the noise $\mathcal{E}_{\theta_j}$ is calculated in the equation 6.2.

[4] It is the most commonly used.

$$
\begin{aligned}
\sigma^2_{\theta_{i+1}} &= E\{\hat{\theta}_{i+1}\hat{\theta}^T_{i+1}\} \text{ where} \\
\hat{\theta}_{i+1} &= \theta_{i+1} - E\{\theta_{i+1}\} = \theta_i + \mathcal{E}_{\theta_1} + \mathcal{E}_{\theta_2} - \theta_i \\
&= \mathcal{E}_{\theta_1} + \mathcal{E}_{\theta_2} \text{ Therefore} \\
\sigma^2_{\theta_{i+1}} &= E\{(\mathcal{E}_{\theta_1} + \mathcal{E}_{\theta_2})(\mathcal{E}_{\theta_1} + \mathcal{E}_{\theta_2})^T\} \\
&= E\{(\mathcal{E}_{\theta_1})^2\} + E\{(\mathcal{E}_{\theta_1})^2\} + \\
&\quad 2E\{(\mathcal{E}_{\theta_1}\mathcal{E}_{\theta_2})\} \text{ where} \\
&\quad E\{(\mathcal{E}_{\theta_1}\mathcal{E}_{\theta_2})\} = 0 \text{ Uncorrelated, and} \\
&\quad E\{(\mathcal{E}_{\theta_1})^2\} = E\{(\mathcal{E}_{\theta_1})^2\} = \sigma_j^2 \text{ Therefore} \\
\sigma^2_{\theta_{i+1}} &= 2\sigma_j^2 \Leftrightarrow \\
\sigma_j &= \frac{\sigma_{\theta_{i+1}}}{\sqrt{2}} \qquad (6.2)
\end{aligned}
$$

More realistically, the translation could be modeled as a series

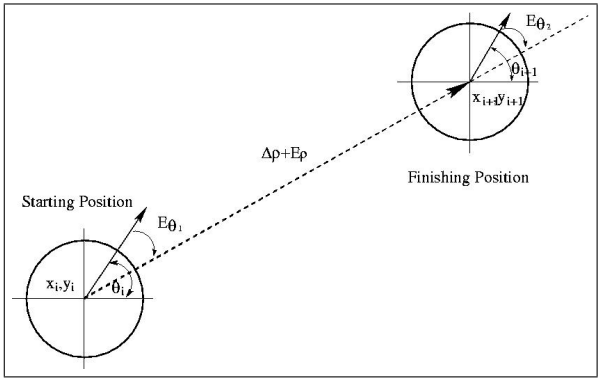


Figure 6.10: One step in translation.

of N equal steps of R/N length each, then the pose of the robot after step i would be: $\overline{\mathbf{x}}_i = (x_i, y_i, \theta_i)^T$ and the trajectory could be modeled as: $\overline{\mathbf{x}}_0, \overline{\mathbf{x}}_1 \ldots \overline{\mathbf{x}}_n$. Figure 6.10 illustrates one step from $\overline{\mathbf{x}}_i$ to $\overline{\mathbf{x}}_{i+1}$. If the translation was performed in one step only then the drift could be modeled as a small rotation before the translation and a small rotation after the translation. Equation 6.3 expresses the above described process, $\mathcal{E}_{\Delta\rho}$ is the noise added in the distance traveled and $\mathcal{E}_{\theta_1}, \mathcal{E}_{\theta_2}$ is the noise added in the orientation of the robot due to drift. The number of steps N used to model the uncertainty should not change the resulting distribution of the robot position. The statistical properties of the distribution of the robot Pose are established experimentally.

$$x_{i+1} = \begin{pmatrix} x_{i+1} \\ y_{i+1} \\ \theta_{i+1} \end{pmatrix} = \begin{pmatrix} x_i + (\Delta\rho + \mathcal{E}_{\Delta\rho}) \cos(\theta_i + \mathcal{E}_{\theta_1}) \\ y_i + (\Delta\rho + \mathcal{E}_{\Delta\rho}) \sin(\theta_i + \mathcal{E}_{\theta_1}) \\ \theta_i + \mathcal{E}_{\theta_1} + \mathcal{E}_{\theta_2} \end{pmatrix} \tag{6.3}$$

Orientation: At the end of step i the orientation of the robot is $\theta_i = \theta_{i-1} + n_i$ where n_i is the noise accumulated during that step. We assume the noise n_i to be zero mean Gaussian and as we saw earlier the result of the addition of two Gaussians (see equation 6.2). Therefore, after the Nth step the orientation of the robot is $\theta_N = \theta_0 + \sum_{i=1}^{N} n_i$. And the statistical properties of the distribution are :

$$\begin{aligned} E\{\theta_N\} &= E\{\theta_0\} + \sum_{i=1}^{N} E\{n_i\} \text{ where} \\ & \sum_{i=1}^{N} E\{n_i\} = 0 \text{ zero mean noise} \Leftrightarrow \\ E\{\theta_N\} &= \theta_0 \end{aligned} \tag{6.4}$$

$$
\begin{aligned}
\sigma_{\mathrm{N}}^2 &= E\{\hat{\theta}_N \hat{\theta}_N^T\} \text{ where } \hat{\theta}_N = \theta_N - E\{\theta_N\} = \theta_N - \theta_1 \Leftrightarrow \\
\sigma_{\mathrm{N}}^2 &= E\{(\theta_N - \theta_1)(\theta_N - \theta_1)^T\} = E\{(\sum_{i=1}^{N} n_i)(\sum_{i=1}^{N} n_i)^T\} \\
&= E\{(n_1 + \ldots + n_N)(n_1 + \ldots + n_{\mathrm{N}})^T\} \\
&= \sum_{i=1}^{N} E\{n_i^2\} + E\{n_1(n_2 + \ldots n_N)^T\} \\
&\quad + E\{n_2(n_1 + n_3 \ldots + n_N)^T\} + \ldots \\
&= \sum_{i=1}^{N} E\{n_i^2\} \\
&\quad \text{because } E\{n_i(\sum_{j=1:N}^{j<>i} (n_j))^T\} = 0 \text{ Uncorrelated } \Leftrightarrow \\
\sigma_{\mathrm{N}}^2 &= N\sigma_i^2 \Leftrightarrow \sigma_{\mathrm{N}} = \sqrt{N}\sigma_i \Leftrightarrow \sigma_{\mathrm{d}rift}\rho = \sqrt{N}\sigma_{step}\frac{\rho}{N} \Leftrightarrow \\
\sigma_{step} &= \sigma_{\mathrm{d}rift} * \sqrt{N} \qquad (6.5)
\end{aligned}
$$

In conclusion, for a given set of uncertainty parameters, defined as $< \sigma_{trans}, \sigma_{drift} >$, the noise $(\mathcal{E}_{\Delta\rho}, \mathcal{E}_{\theta_1}, \mathcal{E}_{\theta_2})$ that should be added during the modeling of odometry error is given in equation 6.6, where $\mathbf{N}(0,1)$ is a random number drawn from a Gaussian distribution with zero mean and sigma equal to one.

$$
\begin{pmatrix} \mathcal{E}_{\Delta\rho} \\ \mathcal{E}_{\theta_1} \\ \mathcal{E}_{\theta_2} \end{pmatrix} = \begin{pmatrix} \mathbf{N}(0,1)\sigma_{trans}\sqrt{N}\Delta\rho \\ \mathbf{N}(0,1)\dfrac{\sigma_{drift}\sqrt{N}\Delta\rho}{\sqrt{2}} \\ \mathbf{N}(0,1)\dfrac{\sigma_{drift}\sqrt{N}\Delta\rho}{\sqrt{2}} \end{pmatrix} \qquad (6.6)
$$

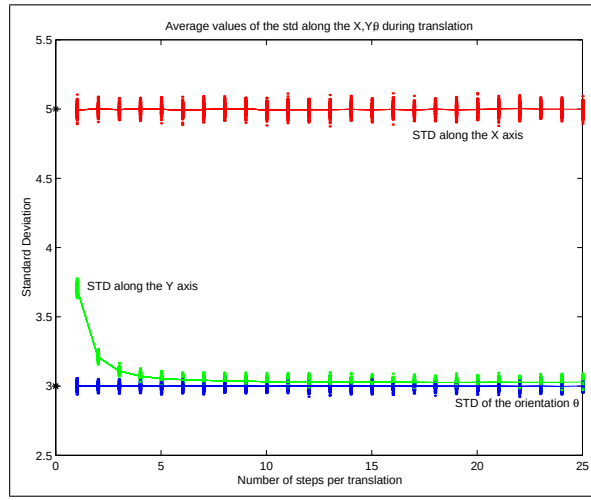


Figure 6.11: The standard deviations of 30000 particles as they move along the x axis for 100cm using different number of steps each time. The experiment is repeated 100 times.

Using the above model we run experiments for different number of steps using multiple samples. It is worth noting that a change in the number of steps affects only the distribution of the points along the direction normal to the direction of the translation and only for small number of steps. As the number of steps increases the standard deviation of the samples along the direction perpendicular to the direction of the translation converges. Figure 6.11 presents the standard deviation of 10000 particles along the X-axis, Y-axis and the orientation after they moved along the X-axis for 300cm, for different number of steps. The standard deviations along the axis of motion and for the orientation is constant for all practical purposes.

Chapter 7

Resampling Methods

In this Chapter three methods of resampling are described together with some variations that help improve the performance. In every case the input is an array [1] of the weights of the particles (normalized to sum up to one) and the output is an array of indices that indicate which particles are going to propagate forward. The premise of all algorithm is that particles with high weights are going to be duplicated while the particles with small (or zero) weights are going to be eliminated.

7.1 Select with Replacement

The simplest method of resampling is to select each particle with a probability equal to its weight. In order to do that efficiently, first the cumulative sum of the particle weights are calculated, and then N sorted random numbers (sorting is $O(n \log(n))$ uniformly distributed in $[0, 1]$ are selected. Finally, the number of the sorted random numbers that appear in each interval of the cumulative sum represents the number of copies of this particular particle which

[1] The arrays start at 1 in MATLAB, 0 in C/C++.

are going to be propagated forward to the next stage. Intuitively, if a particle has a small weight, the equivalent cumulative sum interval is small and therefore, there is only a small chance that any of the random numbers would appear in it; in contrast, if the weight is large then many random numbers are going to be found in it and thus, many duplicates of that particle are going to survive. Algorithm 3 presents a formal description of the "select with replacement" algorithm.

```
  Input: double W[N]
Require: ∑_{i=1}^{N} W_i = 1
  Q = cumsum(W); { calculate the running totals Q_j = ∑_{l=0}^{j} W_l }
  t = rand(N+1); {t is an array of N+1 random numbers.}
  T = sort(t); {Sort them (O(n log n) time)}
  T(N+1) = 1; i=1; j=1; {Arrays start at 1}
  while (i ≤ N) do
    if T[i] < Q[j] then
      Index[i]=j;
      i=i+1;
    else
      j=j+1;
    end if
  end while
  Return(Index)
```

Algorithm 3: Select with Replacement Resampling Algorithm; functions are noted as underlined text, Comments are inside curly brackets "{}".

7.2 Linear time Resampling

Carpenter *et al.* [10] proposed a linear time algorithm for resam-

```
  Input: double W[N]
Require: ∑_{i=1}^{N} W_i = 1
  Q = cumsum(W); {calculate the running totals Q_j = ∑_{l=0}^{j} W_l}
  t = -log(rand(N+1));
  T = cumsum(t); {calculate the running totals T_j = ∑_{l=0}^{j} t_l}
  TN = T/T(N+1);{normalize T to TN;}
  i=1; j=1; {Arrays start at 1}
  while (i ≤ N) do
    if T[i] < Q[j] then
      Index[i]=j;
      i=i+1;
    else
      j=j+1;
    end if
  end while
  Return(Index)
```

Algorithm 4: Linear Time Resampling Algorithm; functions are noted as underlined text, Comments are inside curly brackets "{}".

pling from a set of particles. It is based on a manipulation of the random number sequence in order to achieve a new sorted random number sequence in linear time. Using the cumulative sum of the negative logarithm of N random numbers uniformly distributed in $[0, 1]$, a new sequence of N sorted random number uniformly distributed in $[0, 1]$ is created. The final step is the same as in the previous algorithm where the particles are selected with a probability proportional to their weights. Algorithm 4 presents a formal description of the "select with replacement" algorithm.

7.3 Resampling by Liu *et al.*

Instead of using directly the weights (w_j) of the particles in order to decide which ones are going to be propagated forward, another number a_j can be used, usually a function of the particles weights ($a_j = f(w_j)$). A generic choice is the the square root ($f(w_j) = \sqrt{w_j}$). Then the new weights (a_j) are normalized so they sum up to the number of particles N ($\sum_{i=1}^{N} a_i = N$). Then each particle is examined separately, and, if its weight (a_j) is greater or equal to one, k copies of it are propagated forward ($k = \lfloor a_j \rfloor$); otherwise, the particle "survives" with probability equal to a_j. One drawback of this approach is that the number of particles after resampling is not N anymore as the choice of how many particles survive is stochastic [2].

7.4 Variations on Resampling

Two main variations at the resampling stage have been proposed: corrective resampling and keeping a small percentage of particles from the old distribution.

7.4.1 Corrective Resampling

Jensfelt *et al.* [27] suggested a modification to the traditional SIR filter that "boosts" the contribution of the sensing versus the contribution of the predictive model. The particle population is "injected" during the update phase with a small number of particles created directly from the sensor data independently of where the rest of the particles are located.

[2] Stochastic is a process that is random but it follows certain distributions.

7.4.2 Maintaining the variance of the distribution

Contrasting to the previous approach is the method of maintaining a small percentage of the particle population independently of their weights. More precisely during the resampling stage a small number of particles selected uniformly from the particle population are being propagated forward given a small weight. The intuition behind this approach is to maintain the coverage of the predictive model in the particle population without affecting the accuracy of the localization.

```
Input: double W[N]
for (j = 1 to N) do {Update the weights}
  a[j] = sqrt(W[j])
end for
sum = 0;
for (j = 1 to N) do {calculate sum_{i=1}^{N} a_i}
  sum = sum + a[i];
end for
for (j = 1 to N) do {Normalize the weights (a) to sum up
to N}
  a[j] = N * a[j]/sum
end for
i=1;
for (j = 1 to N) do {For each particle}
  if (a[j] ≥ 1) then {Accept the ones with bigger
  weights}
    for (l = 1 to ⌊a[j]⌋ ) do {Add ⌊a_j⌋ copies of the j^th
    Particle}
      Index[i]=j;
      i = i + 1;
    end for
  else
    R =rand(1);
    if a[j] ≥ R then {Accept the particle with
    probability a_j}
      Index[i]=j;
      i = i + 1;
    end if
  end if
end for
Return(Index)
```

Algorithm 5: Resampling Algorithm; functions are noted as underlined text, Comments are inside curly brackets "{}".

Bibliography

[1] Ivan A. Bachelder and Allen M. Waxman. A view-based neurocomputational system for relational map-making and navigation in visual environments. *Robotics and Autonomous Systems*, 16:267–289, 1995.

[2] M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Trans. on Robotics and Automation*, 13(2):251–263, April 1997.

[3] Michael Black and David Fleet. Probabilistic detection and tracking of motion boundaries. *International Journal of Computer Vision*, 38(3):231–245, 2000.

[4] J. Borenstein. The clapper: a dual-drive mobile robot with internal correction of dead-reckoning errors. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3085–3090, San Diego, CA,, May 8-13 1994.

[5] J. Borenstein, H. R. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques.* Number ISBN 1-56881-058-X. A K Peters, Wellesley, MA, 1996.

[6] J. Borenstein, H R. Everett, L. Feng, and D. Wehe. Mobile robot positioning: sensors and techniques. *Journal of Robotic Systems*, 14(4):231–249, Apr 1997.

[7] Johann. Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics & Automation*, 12(6):869–880, Dec 1996.

[8] S. M. Bozic. *Digital and Kalman filtering.* Edward Arnold, second edition, 1994.

[9] Wolfram Burgard, Dieter Fox, Mark Moors, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 476 –481, San Francisco, CA, May 2000. IEEE Press.

[10] J. Carpenter, P. Clifford, and P. Fearnhead. An improved particle filter for non-linear problems. *IEE proceedings - Radar, Sonar and Navigation*, 146:2–7, 1999.

[11] Kok Seng Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, volume 4, pages 2783 –2788, 1997.

[12] A. Curran and K J. Kyriakopoulos. Sensor-based self-localization for wheeled mobile robots. *Journal of Robotic Systems*, 12(3):163–176, Mar 1995.

[13] Frank Dellaert, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* IEEE Press, June 1999.

[14] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[15] Frank Dellaert and Ashley Stroupe. Linear 2d localization and mapping for single and multiple robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2002*. IEEE, May 2002.

[16] Arnaud Doucet, Nando De Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, Series Statistics for Engineering and Information Science, January 2001.

[17] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Number ISBN: 0521568765. Cambridge University Press, May 2000.

[18] Gregory Dudek and Chi Zhang. Vision-based robot localization without explicit object models. In *Proc. International Conference of Robotics and Automation*, Minneapolis, MN, 1996. IEEE Press.

[19] L. Feng, J. Borenstein, and H.R. Everett. Where am i: Sensors and methods for mobile robot positioning,. Technical Report UM-MEAM-94-21, University of Michigan, Ann Arbor, University of Michigan, Ann Arbor, MI, USA, December 1994.

[20] Toshio Fukuda, Shigenori Ito, Fumihito Arai, Yasunari Yokoyama, Yasunori Abe, Kouetsu Tanaka, and Yoshio Tanaka. Navigation system based on ceiling landmark recognition for autonomous mobile robot - landmark detection based on fuzzy template matching (ftm). In *IEEE International Conference on Intelligent Robots and Systems*, volume 2, pages 150–155, 1995.

[21] Arthur Gelb. *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts, 1974.

[22] F. Giuffrida, C. Massucco, P. Morasso, G. Vercelli, and R. Zaccaria. Multi-level navigation using active localization system.

In *IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 413–418. IEEE, 1995.

[23] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings For Radar and Signal Processing*, 140(2):107 –113, April 1993.

[24] Robert Grabowski and Pradeep Khosla. Localization techniques for a team of small robots. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, number ISBN:0-7803-6614-X, pages 1067–1072, Maui, Hawaii, USA, Oct. 29 - Nov. 03 2001.

[25] Michael Isard and Andrew Blake. Condensation-conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):2–28, 1998.

[26] Michael Isard and Andrew Blake. Icondensation: Unifying low-level and high-level tracking in a stochastic framework. In *Proc 5th European Conf. Computer Vision*, volume 1, pages 893–908, 1998.

[27] Patric Jensfelt, Olle Wijk, David J. Austin, and Magnus Andersso. Experiments on augmenting condensation for mobile robot localization. In *IEEE International Conference on Robotics & Automation (ICRA)*, pages 2518–2524, San Francisco, CA, USA, April 2000.

[28] Patric Jensfelt, Olle Wijk, David J. Austin, and Magnus Andersso. Feature based condensation for mobile robot localization. In *IEEE International Conference on Robotics & Automation (ICRA)*, pages 2531–2537, San Francisco, CA, USA, April 2000.

[29] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[30] S. Koenig and R.G. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2301–2308. IEEE, 1996.

[31] R. Kurazume and S. Hirose. Study on cooperative positioning system - optimum moving strategies for cps-iii. In IEEE, editor, *Proc. IEEE Int. Conf. on Robotics and Automation*, volume 4, pages 2896–2903, 1998.

[32] Ryo Kurazume, Shigeo Hirose, Shigemi Nagata, and Naoki Sashida. Study on cooperative positioning system. In *International Conference in Robotics and Automation*, volume 2, pages 1421–1426. IEEE, April 1996.

[33] Ryo Kurazume and Shigemi Nagata. Cooperative positioning with multiple robots. In *International Conference in Robotics and Automation*, volume 2, pages 1250–1257. IEEE, 1994.

[34] John J. Leonard and Hugh F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.

[35] C. Lin and R. Tummala. Mobile robot navigation using artificial landmarks. *Journal of Robotic Systems*, 14(2):93–106, 1997.

[36] Jun S. Liu, Rong Chen, and Tanya Logvinenko. A theoretical framework for sequential importance sampling and resampling. In A. Doucet, N. de Freitas, and N.J. Gordon, editors, *Sequential Monte Carlo in Practice*. Springer-Verlag, January 2001.

[37] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[38] Feng Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, pages 249–275, 1998.

[39] John MacCormick and Andrew Blake. A probabilistic exclusion principle for tracking multiple objects. In *Proc. Int. Conf. Computer Vision*, pages 572–578, 1999.

[40] Paul MacKenzie and Gregory Dudek. Precise positioning using model-based maps. In *Proceedings of the International Conference on Robotics and Automation*, San Diego, CA, 1994. IEEE Press.

[41] P. Maybeck. *Stochastic Models, Estimation ond Control*, volume 1. Academic, New York, 1979.

[42] Jong-Woo Moon, Chong-Kug Park, and Fumio Harashima. Kinematic correction of a differential drive mobile robot and a design for velocity trajectory with acceleration constraints on motor controllers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 930–935, Piscataway, NJ, USA, 1999.

[43] F. Nashashibi and M. Devy. Combining terrain maps and polyhedral models for robot navigation. In *1993 International Conference on Intelligent Robots and Systems.*, pages 685–691, July 1993.

[44] Ioannis Rekleitis, Gregory Dudek, and Evangelos Milios. Probabilistic cooperative localization and mapping in practice. In *IEEE International Conference on Robotics and Automation*, pages 1907–1912, Taipei, Taiwan, 2003. IEEE.

[45] Ioannis M. Rekleitis. *Cooperative Localization and Multi-Robot Exploration.* PhD thesis, School of Computer Science, McGill University, Montreal, Quebec, Canada, February 2003.

[46] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Québec, CANADA H3A 2A7, 2004.

[47] Ioannis M. Rekleitis, Gregory Dudek, and Evangelos Milios. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):7–40, 2001.

[48] Stergios I. Roumeliotis and George A. Bekey. Bayesian estimation and kalman filtering: A unified framework for mobile robot localization. In *Proc. 2000 IEEE International Conference on Robotics and Automation*, pages 2985–2992, San Francisco, California, April 22-28 2000. IEEE.

[49] Stergios I. Roumeliotis and George A. Bekey. Collective localization: A distributed kalman filter approach to localization of groups of mobile robots. In *Proc. 2000 IEEE International Conference on Robotics and Automation*, pages 2958–2965, San Francisco, California, April 22-28 2000. IEEE.

[50] Stergios I. Roumeliotis and George A. Bekey. Distributed multi-robot localization. In *5th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 179–188, Knoxville, Tennessee, USA, October 4-6 2000. Springer.

[51] N. Roy and S. Thrun. Online self-calibration for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[52] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, Winter 1986.

[53] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I.J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.

[54] J. Sullivan, A. Blake, M. Isard, and J. MacCormick. Object localization by Bayesian correlation. In *Proc. Int. Conf. Computer Vision*, pages 1068–1075, 1999.

[55] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence Journal*, 101:99–141, 2001.

[56] Nikos Vlasis, Bas Terwijn, and Ben Krose. Auxiliary particle filter robot localization from high-dimensional sensor observations. In IEEE, editor, *IEEE International Conference in Robotics and Automation*, volume 1, pages 7–12, May 2002.

[57] Gerhard Weiss, Christopher Wetzler, and Ewald. von Puttkamer. Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In *IEEE/RSJ/GI International Conference on Intelligent Robots and Systems.*, volume 1, pages 595–601. IEEE, 1994.

Appendix

Piece Wise Linear Motion Sample Code

Init

```
function pos=init(x,y,theta,N)
% Usage: ;
% pos=init(x,y,theta,N);
% Initializes a set of "N" pose particles;
% pos: (an N by 4 matrix) at the pose <x,y,theta>.

     pos=zeros(N,4);
     pos(1:N,1)=x;
     pos(1:N,2)=y;
     pos(1:N,3)=theta;
     pos(1:N,4)=1/N;
```

RelRotate

```
function Pose=RelRotate(theta,Pose)
% Rotates an array of particles by theta,
% Adds error of N(rotMean,rotSigma*Dtheta/360) where :
%   -rotSigma (global variable) is the std of
%    rotation in deg per 360.
%   -rotMean (global variable) is the mean in
%    degrees (usually zero)
%   -Dtheta is the individual rotation needed for
%    each particle for a rotation by relative theta
%    degrees.

     N=size(Pose,1);
     global rotSigma;
     Err=randn(N,1).*theta*rotSigma/360;
     Pose(1:N,3)=Pose(1:N,3)+theta+Err;
```

Translate

```
function NewPose=Translate(r,Pose)
     % Translate a set of particles (Pose) forward by r;
     N=size(Pose,1); piconst=pi/180.0;

     global trsSigma;
     global drfSigma;
     global trsMean;
     global drfMean;
     global Nstps;  % Number of steps
     if(Nstps<1) Nstps=1; end;
     stepTrans=r/Nstps;
     ltrsSigma = trsSigma*sqrt(Nstps)*stepTrans/100; % per cent

     % Local drift sigma (in degrees per meter);
     ltrsRotSigma= drfSigma*sqrt(Nstps)*stepTrans/(100*sqrt(2));
     for(k=1:1:Nstps)
        Err1=randn(N,1)*ltrsSigma+trsMean*stepTrans/100;
        Err2=randn(N,1)*ltrsRotSigma;

        Pose(1:N,3)=Pose(1:N,3)+Err2;
        Angle=Pose(1:N,3)*piconst;
        Pose(1:N,1)=Pose(1:N,1)+(stepTrans +Err1).*cos(Angle);
        Pose(1:N,2)=Pose(1:N,2)+(stepTrans +Err1).*sin(Angle);

        Err2=randn(N,1)*ltrsRotSigma+drfMean*stepTrans/100;

        Pose(1:N,3)=Pose(1:N,3)+Err2;

     end;
     NewPose=Pose;
```

Velocity Control Propagation code

move

```
function P=move(v,w,Pos)
 % Move a set of particles (Pos) by linear velocity v
 % and angular velocity w for 1 sec.
 N=size(Pos,1);
 P=zeros(N,4);
 dt=1;
 sigma_v=0.03;
 sigma_w=0.002;
 e_v=randn(N,1)*sigma_v;
 e_w=randn(N,1)*sigma_w;
 P(:,1)=Pos(:,1)+(v+e_v).*dt.*cos(Pos(:,3));
 P(:,2)=Pos(:,2)+(v+e_v).*dt.*sin(Pos(:,3));
 P(:,3)=Pos(:,3)+(w+e_w).*dt;
 P(:,4)=Pos(:,4);
```

Init

Init

Faulty Commands example

```
function testFaultyCommands(N)
%Piecewise linear motion (Translation and Rotation)
%Command success 70%; Start at [-8,0,0];
% Translate by 4m; Rotate by 30 degrees; Translate by 6m
P=init(-8,0,0,N);
```

```
close all; f=figure; % plot coordinate system
plot([-10 4],[0,0],'k'); hold on; plot([0 0],[-4, 10],'k');

% plot Original Pose
plot(P(:,1),P(:,2),'r.'); title('Original Pose');
saveas(f,'faultyComm1.pdf');

%Select ~70% to receive/use a command: Translate by 4 m.
idx=rand(N,1);
stay=P(find(idx>0.7),:); move=P(find(idx<=0.7),:);
move=Translate(4,move); P=[move;stay];
plot(P(:,1),P(:,2),'r.'); title('Translate by 4 m');
saveas(f,'faultyComm2.pdf');

%Select ~70% to receive/use a command: Rotate by 30 degrees.
idx=rand(N,1);
stay=P(find(idx>0.7),:); move=P(find(idx<=0.7),:);
move=RelRotate(30,move); P=[move;stay];
plot(P(:,1),P(:,2),'r.'); title('Rotate by 30 degrees');
saveas(f,'faultyComm3.pdf');

%Select ~70% to receive/use a command: Translate by 6 m.
idx=rand(N,1);
stay=P(find(idx>0.7),:); move=P(find(idx<=0.7),:);
move=Translate(6,move); P=[move;stay];
plot(P(:,1),P(:,2),'r.'); title('Translate by 6 m');
saveas(f,'faultyComm4.pdf');
```