A Comparison of Mobile Robot Pose Estimation using Non-linear Filters: Simulation and Experimental Results

Zongwen Xue and Howard Schwartz
Department of System and Computer Engineering
Carleton University
1125 Colonel By Drive, Ottawa, ON, Canada K1S 5B6

Abstract—This paper explores and compares the nature of the non-linear filtering techniques on mobile robot pose estimation. Three non-linear filters are implemented including the extended Kalman filter (EKF), the unscented Kalman filter (UKF) and the particle filter (PF). The criteria of comparison is the magnitude of the error of pose estimation, the computational time, and the robustness of each filter to noise. The filters are applied to two applications including the pose estimation of a two-wheeled robot in an experimental platform and the pose estimation of a three-wheeled robot in a simulated environment. The robots both in the experimental and simulated platform move along a non-linear trajectory like a circular arc or a spiral. The performance of their pose estimation are compared and analysed in this paper.

Index Terms—extended Kalman filtering; unscented Kalman filtering; particle filtering; Monte Carlo Methods; mobile robot tracking; Pose estimation.

1 Introduction

In order to fulfill the desired tasks, mobile robots are required to localize themselves precisely. Generally, pose estimation is referred to as the ability to estimate the position and orientation of a mobile robot from noisy measurements. The Extended Kalman Filter (EKF) [1] is a standard approach to perform data fusion and state estimation.

The EKF is based on Taylor-series expansion of nonlinear system functions, and then expectations are used to compute and update mean and covariance of state distribution [2], [3]. However, given the high non-linearity of the system, the EKF is prone to introduce large errors or even diverge [3].

The UKF [7] was proposed on the assumption that it is generally easier to approximate the state distribution of a Gaussian random variable (GRV) than to locally linearize a non-linear function. It implicitly captures the first and second moments of the state distribution of a GRV [8].

However, the analytical approaches such as the UKF and the EKF are not applicable to the non-Gaussian filtering problems [9]. The Sequential Monte Carlo (SMC) filter is an alternative approach and can improve performance of state estimate. The SMC-based method is also referred to as the particle filter (PF). The PF is to approximate the posterior PDF directly by a large set of random sample points (particles). The particles drawn from a known PDF are propagated through the system model. The true statistical distribution of the state vector can be estimated by means of the particles and their importance weight [10].

Many studies have been conducted to compare the nature of the EKF, UKF and the PF for trajectory tracking and position estimation [4]–[6]. The results of our experiments and simulations provide the additional evidences that:

1) The PF has outperforms the EKF and the UKF in terms of accuracy and robustness on the highly non-linear systems with the non-Gaussian noise.
2) The UKF has almost the same level of accuracy to the EKF but it needs more computational efforts.
3) The PF has the most computational cost. Thus it it often not practical for the real time applications.

This paper is structured as follows: Section 2.1 presents the model of the mobile robot and the measurement model in the simulation. The architecture of the experiment and the model of a two-wheeled robot is described in Section 2.2. Section 3 introduces the theories and implementations of underlying non-linear filtering approaches including the EKF, UKF and the PF. Finally, the results and the numerical analysis of the simulation and experiments are illustrated and explained in Section 4.

2 Applications to Mobile Robotics

2.1 A Simulated Environment

2.1.1 System Model of Three-Wheeled Robot

In the simulations, a three-wheeled vehicle mobile robot is simulated as shown in Fig. 1. The model of the mobile robot is depicted in Fig. 2. It is assumed that the robot moves on a 2D horizontal plane. The robot model has been derived assuming no slippage in [11]. The robot is driven by its rear wheels and the front wheel is a steering wheel. One refers to the combination of position and orientation as the pose of a robot. Thus the system state of the pose of the robot is defined as $X_k = (x_k, y_k, \theta_k)^T$ where x_k and y_k specify the coordinates of the point M in the Cartesian plane in Fig. 2. The point M is mid-distance between rear wheels of the robot. The term θ_k denotes the robot tuning angle with respect to x-axis of the Cartesian plane. The term δ is the turning angle of the front wheel with respect to the robot's longitudinal orientation.

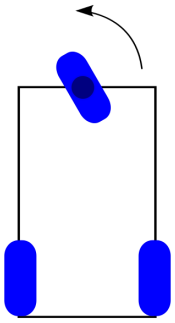

Fig. 1. Three wheeled omni-steer mobile robot

The model of the robot is described by the following stochastic differential equations (SDE) [12]:

$$\dot{X}(t) = f(X) + \dot{w} \tag{1}$$

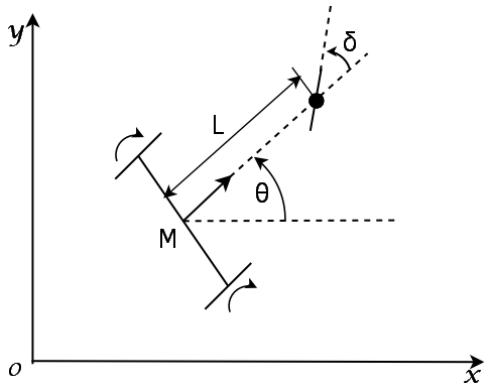

Fig. 2. Model of Mobile Robot

where $f(\cdot)$ is a nonlinear system function. The term w specifies an additive zero-mean random process noise. The differential equation of the state vector for the robot model has the form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{L}\tan\delta \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \\ w_\theta \end{bmatrix} \tag{2}$$

where v specifies the robot longitudinal velocity, and $\dot{\theta}$ is the chassis instantaneous angular velocity. The term δ characterizes the vehicle's steering wheel angle, and L denotes the distance between the rear and front wheels' axles. In all forthcoming simulations, L is set equal to 5 cm.

The process noise w in the simulations is a zero mean Gaussian random variable(GRV). It includes all kinds of errors that can not be detected by odometers such as range error, turn error and drift error. Moreover, the process noise w is independent of the system states. The standard deviations of the process noise w in all the simulations are assumed to be $\sigma_x = \sigma_y = 1$cm and $\sigma_\theta = \frac{\pi}{180}$rad. The covariance matrix of the process noise can be defined as: $Q = E[ww^T]$ where $w_x \sim \mathcal{N}(0, \sigma_x^2)$, $w_y \sim \mathcal{N}(0, \sigma_y^2)$ and $w_\theta \sim \mathcal{N}(0, \sigma_\theta^2)$.

2.1.2 Measurement Model

Three measurements are available at each time step as shown in Fig. 3. The range r_1 and r_2 are the measurements of the distances from the robot to the landmark 1 and landmark 2. The bearing α represents the angular measurement between the mobile robot's longitude and the line from the robot to the landmark 3.

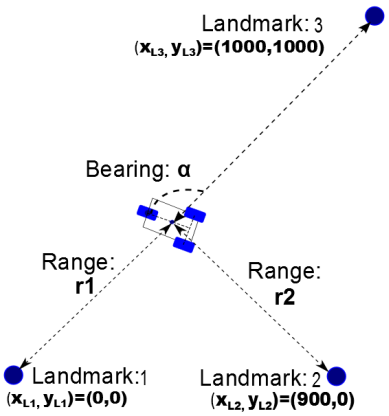

Fig. 3. Model of Measurement System

$$z_k = \begin{bmatrix} r_{1k} \\ r_{2k} \\ \alpha_k \end{bmatrix} = h(x_k) + \eta_k$$
$$= \begin{bmatrix} \sqrt{(x_k - x_{L1})^2 + (y_k - y_{L1})^2} \\ \sqrt{(x_k - x_{L2})^2 + (y_k - y_{L2})^2} \\ \arctan(\frac{y_{L3} - y_k}{x_{L3} - x_k}) - \theta_k \end{bmatrix} + \begin{bmatrix} \eta_{r1} \\ \eta_{r2} \\ \eta_\alpha \end{bmatrix} \quad (3)$$

where (x_{L1}, y_{L1}), (x_{L2}, y_{L2}) and (x_{L3}, y_{L3}) specify the Cartesian coordinates of the landmarks 1, 2 and 3 respectively. The terms x_k and y_k denote the Cartesian coordinates of the robot location. The terms η_{r1}, η_{r2} and η_α represent additive zero-mean Gaussian noise of the measurements. The standard deviations of the measurement noise are σ_{r1}, σ_{r2} and σ_α with respect to the measurements of $r1$, $r2$ and α. Thus $\eta_{r1} \sim \mathcal{N}(0, \sigma_{r1}^2)$, $\eta_{r2} \sim \mathcal{N}(0, \sigma_{r2}^2)$ and $\eta_\alpha \sim \mathcal{N}(0, \sigma_\alpha^2)$. We assume that the standard deviations of the measurement errors are $\sigma_{r1} = 5$cm, $\sigma_{r2} = 5$cm and $\sigma_\alpha = \frac{\pi}{180}$rad in the simulations. The measurements are taken at a frequency of 5 Hz. That is, the sample period is $T_s = 0.2s$.

2.2 Experimental Platform

A robot experimental platform is necessary and useful at evaluating the performance of pose estimation. An experimental platform which is called the Mobile Cooperating Robot Test Platform(MCRTP) is utilized in the experiments to perform mobile robot pose estimation. The MCRTP is a low cost, scalable, cooperative robot test system. It consists of the Arduino-based differential robots and a camera system. The camera system is a combination of a USB digital camera and a software application. It can track and broadcast robot positions over a wireless network.

2.2.1 Architecture

The MCRTP has a client-server architecture. The camera system and a computer are classified as the server. One or more mobile robots are thought of as the remote workstations. The camera system consists of a USB digital camera and a software application running on a computer. The camera is fixed at the ceiling on the top of the experimental area. It is connected to the computer with a USB cable. The Arduino-based mobile robot embeds a XBee receiver which allows its actuator and sensors to be accessed remotely through a wireless network. Fig. 4 presents the architecture of the experimental platform. The mobile robots in the

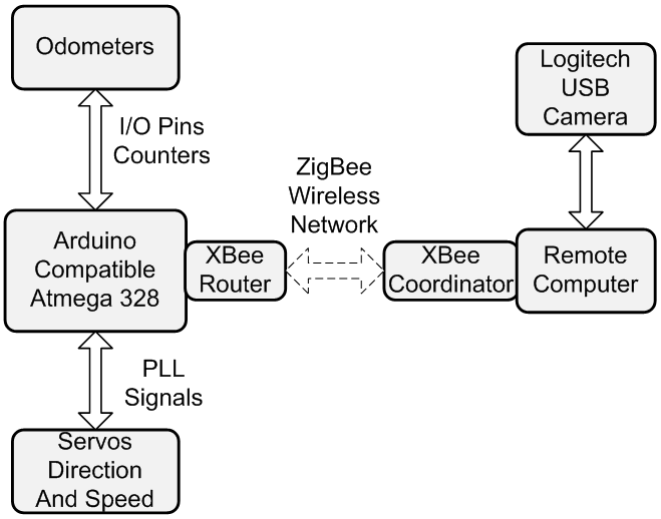

Fig. 4. Architecture of the Experimental Platform

MCRTP is controlled by *Romeo-All in one Controller*, an Arduino compatible microcontroller. The robots as shown in Fig. 5 are small, two-wheeled, battery powered robots. The key hardware components of the robot are the Arduino microcontroller, the motor shield, and the XBee chip. Arduino is an open source computing platform designed specifically for running external devices such as motors and sensors [13]. The Arduino compatible board uses an Atmega328 microcontroller. The board was powered by one 7.4v LiPo(1600mAh) rechargeable battery and the motors (298:1 Micro Metal Gear Motor) were rated at 6V with a 298:1 gear ratio. The computer exchanges data with the mobile robots through a ZigBee wireless network as illustrated in Fig. 4.

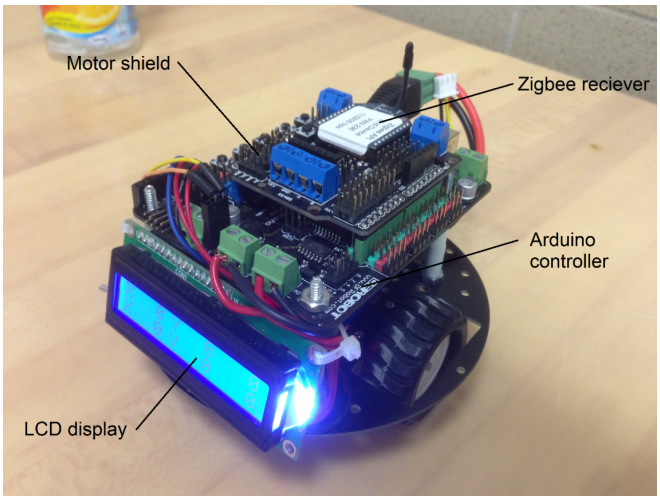

Fig. 5. A 2-wheeled mobile robot with a font-facing LCD display.

In order to track the motions of the mobile robots, a software application which is called the vision system is running on the computer of the experimental platform. The vision system is developed in C++ and it is running on the Windows operating systems. The vision system makes use of a USB camera to locate the robots and communicates with the robots through a XBee transmitter. The USB camera (Logitech HDc615) records images and visually tracks the positions of the robots.

In order to identify the robot, the recorded images are processed by another software application, the *Open Computer Vision Library* (OpenCV) [14]. The OpenCV software library can filter images by colours. The colours are identified by the OpenCV software library with respect to a HSV (hue, saturation, value) value. Each robot is equipped with a rectangular colour identifier. The OpenCV software library employs an algorithm, namely, the *Canny edge detection* algorithm to identify the rectangular color identifier which is attached on top of the robot. The positions of the robot are identified individually with respect to its colour identifier by the OpenCV software library. A screenshot of OpenCV vision system is presented in Fig.6.

The computer running the MCRTP system has a 3.3 GHz Intel Core i3 processor and 4G RAM. The operating system is Windows 7 32-bit. It has enough computing power to efficiently implement all tasks including all the software tasks and the external hardware tasks in the experiments.

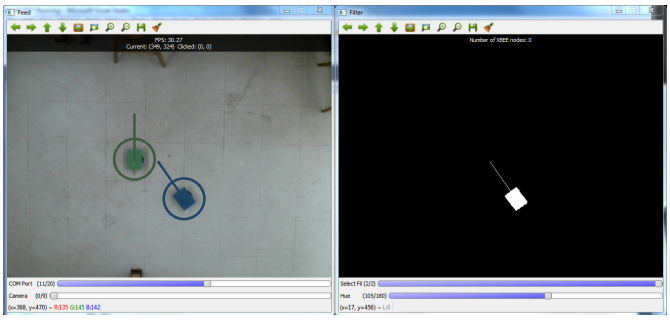

Fig. 6. Screenshot of the OpenCV vision system, On the left robots is identified by colour. On the right is the filtered view for an individual colour.

2.2.2 *The Mobile Robot and Its Actuator*

The MCRTP uses Arduino-based autonomous robots in the experiments. The differential two wheeled robots are steered by spinning the wheels at different speeds with respect to one another. The left and right wheels of a robot are driven by two different motors. The motor's speed is controlled by the digital input which is a duty cycle value written to the motor pins. The duty cycle values are scalars between 0 and 255. Each duty cycle scalar is relative to 127 (both wheels are stopped), and scalars less than 127 spin the wheel backwards, and scalars greater than 127 spin the wheel forwards.

Notice that the robots are running an open loop system in the experiments. That is, there is no feedback of the system states to reflect the effect of the programmed actions. For example, the robot may not run in a straight path even though the same digital inputs are applied to the motors that drive the left and the right wheels. This of course is the expected result due to electro-mechanic difference which is modelled as part of the process noise.

There are many factors that contribute to the process noise including the battery strength, the traction of the surface, the difference between the left and the right wheels of the robots. Moreover, two motors on each mobile robot have different levels of noise and cannot produce exactly the same torques even if they are applied to the same inputs(the same voltage). First the sizes of the robot wheels of the robots are usually slightly different from the manufacturer standard. These would cause calculation errors of the robot's speed. Second, since the sizes of the two wheels on each robot cannot be the same exactly, it leads to a calculation error of the robot's turning angle. The wheels of a robot also may slip on the ground during the

experiment. Finally, battery strength is probably one of the most significant factors determining the torque on the wheels. All the above errors may lead to inconsistent results of the robot's actual displacements.

The mobile robots are equipped with gear-motors with $298:1$ metal gearbox which has a Free-Run speed of $100RPM$ at $6V$. The circumference of robot wheels is $131.95mm$. Thus the maximum speed of a mobile robot along a straight path can be calculated as $0.7215ft/sec$ and $22.0cm/sec$. A sample interval $T_0 \leq 100ms$ (corresponding to a displacement of less than $3.0cm$) would be reasonable.

After conducting some tests to evaluate the process noise, the result indicates the process error is about 5% of the total length of the trajectory. We finally determine that the process error in our experiments can be set $\sigma_{v_L} = \sigma_{v_R} = 1.0cm \approx 2.5$ pixels. We also assume that all process errors are additive Gaussian white noise.

2.2.3 System Model of the Differential Drive Robot

In order to perform pose estimation with the nonlinear filters, the system dynamic model of the mobile robots is required. The derivation of the system dynamic model is shown from Eqn.4 to Eqn.17. In Fig.7, the term R denotes the instantaneous curvature radius of the robot trajectory with relation to the center of the axis. The term l specifies the distance between axles of the two wheels. In our case, the length l is equal to 3.7 cm. Let v_R and v_L be the linear velocities of the right and left wheel respectively. Analytically, these linear velocities can be expressed as

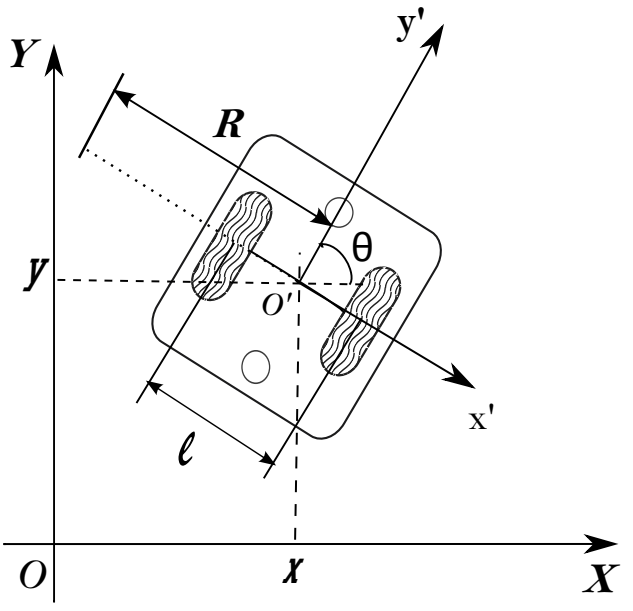

Fig. 7. Kinematic Model of a Differential Drive Mobile Robot.

$$v_R = \left(R + \frac{l}{2}\right)\omega \tag{4}$$

$$v_L = \left(R - \frac{l}{2}\right)\omega \tag{5}$$

where ω is the the angular velocity of the robot. Solving for ω yields:

$$\omega = \frac{v_R}{R + \frac{l}{2}} \tag{6}$$

$$\omega = \frac{v_L}{R - \frac{l}{2}} \tag{7}$$

and set Eqn. (6) equal to Eqn. (7):

$$\begin{aligned} & \frac{v_R}{R+\frac{l}{2}} = \frac{v_L}{R-\frac{l}{2}} \\ \Rightarrow \quad & v_R\left(R - \frac{l}{2}\right) = v_L\left(R + \frac{l}{2}\right) \\ & R(v_R - v_L) = \frac{l}{2}(v_R + v_L) \\ & R = \frac{l}{2} \times \frac{v_R + v_L}{v_R - v_L} \end{aligned} \tag{8}$$

substituting Eqn. (8) back into Eqn.(6) yields:

$$\begin{aligned} \omega &= \frac{v_R}{\left(\frac{l}{2} \times \frac{v_R + v_L}{v_R - v_L}\right) + \frac{l}{2}} \\ &= \frac{v_R}{\frac{l}{2}\left(\frac{v_R + v_L}{v_R - v_L} + \frac{v_R - v_L}{v_R + v_L}\right)} \\ \omega &= \frac{1}{l}v_R - \frac{1}{l}v_L \end{aligned} \tag{9}$$

Finally, substituting (8) and (9) into $v = R\omega$ yields:

$$v = \frac{1}{2}(v_R + v_L) \tag{10}$$

The stochastic differential equations of the kinematic model of the differential drive mobile robots are:

$$\dot{X} = f(X) + w \tag{11}$$

$$= \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \\ w_\theta \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2}(\upsilon_R + \upsilon_L)\cos\theta \\ \frac{1}{2}(\upsilon_R + \upsilon_L)\sin\theta \\ \frac{1}{l}\upsilon_R - \frac{1}{l}\upsilon_L \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \\ w_\theta \end{bmatrix} \tag{12}$$

$$\dot{X} = \begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \upsilon_R \\ \upsilon_L \end{bmatrix} \tag{13}$$

$$+ \begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} w_{\upsilon_R} \\ w_{\upsilon_L} \end{bmatrix} \tag{14}$$

where

$$w_x = \frac{1}{2}\cos\theta(w_{\upsilon_R} + w_{\upsilon_L}) \tag{15}$$

$$w_y = \frac{1}{2}\sin\theta(w_{\upsilon_R} + w_{\upsilon_L}) \tag{16}$$

$$w_\theta = \frac{1}{l}(w_{\upsilon_R} - w_{\upsilon_L}) \tag{17}$$

2.2.4 The Sensor of The Measurement System

The measurements in the experiments are of the state directly. Therefore, the measurement model has the form:

$$z_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \eta_x \\ \eta_y \\ \eta_\theta \end{bmatrix} \tag{18}$$

As described in Sec.2.2.1, the task of the camera system in the MCRTP is to track the mobile robots. The camera system consists of a digital camera and a software application. The digital camera is a Logitech HDc615 webcam set which is capable of recording images with a 640×480 frame size. The digital images can be captured and sent back to the computer periodically during the experiment. The images are processed by the OpenCV to generate the different filtered images with respect to the robot's coloured identifier. Finally, the mobile robot's real-time positions and orientations are calculated.

In order to evaluate the accuracy of the camera system, we conducted another experiment by comparing the real measurements of a tape with the readings from the camera system. The experiment was conducted by dividing the experimental area into $35(5 \times 7)$ squares and each has the same size (1 foot in square). The pixels in x and y directions of each square are recorded. If we assume the square in the middle of frame is perfect and without distortion, the maximum error can be calculated by comparing the difference of the number of pixels in x and y directions of each square with the number of pixels of the assumed perfect square.

The maximum error in x-direction yielded from the experiments is about ± 9 pixels (corresponding to $3.5cm$). Whereas the maximum error in y-direction is about ± 3 pixels (corresponding to $1.2cm$). Thus the total error yielded in the experiment would be less than $4cm$ and the camera has an accumulative positional accuracy of over 96%. In this case, we can set the standard deviation of the measurement errors σ_x and σ_y with respect to the movements along the x and y directions as $\sigma_x = 4cm \approx 10$ pixels and $\sigma_y = 1.2cm \approx 3.0$ pixels. In the experiments, we noticed that the noise of the camera system is relatively small. Moreover, most of the measurement errors are produced by the distortion of the lens which has non-Gaussian distribution. In order to compare the performance of the filters on the basis of the measurements disturbed by the noise with zero mean and Gaussian distribution, white noise is deliberately added into each measurement in the experiments as follows $w_x \sim \mathcal{N}(0, \sigma_x^2)$, $w_y \sim \mathcal{N}(0, \sigma_y^2)$ and $w_\theta \sim \mathcal{N}(0, \sigma_\theta^2)$ where $\sigma_x = 10$ pixels, $\sigma_y = 10$ pixels and $\sigma_\theta = 5\pi/180$.

3 Nonlinear Filtering Algorithms For Mobile Robot Pose Estimation

3.1 Extended Kalman Filtering

In the implementation of the EKF, the system model $f(X)$ can be represented by a set of first-order nonlinear differential equations:

$$\dot{X} = f(X) + w \tag{19}$$

where $\dot{X} = [\dot{x}, \dot{y}, \dot{\theta}]^T$ characterizes the system states. The term $f(X)$ is a set of non-linear differential equations, and w defines a matrix representing the random process noise. The process noise w_k is a zero-mean GRV, and its covariance matrix can be defined as:

$$w_k = \begin{bmatrix} w_x \\ w_y \\ w_\theta \end{bmatrix} \qquad Q_k = E[w_k w_k^T] \tag{20}$$

Thus the process noise covariance matrix is given as $Q_k = diag(w_x^2, w_y^2, w_\theta^2)$.

The measurement model normally is given by $z = h(x) + \eta$. The term η is a zero-mean random measurement error. The measurement model can also be written in discrete form as $z_k = h(x_k) + \eta_k$. The covariance matrix of the measurement noise R is defined as $R = E(\eta\eta^T)$. The discrete form of the measurement error covariance is given by $R_k = diag\{\eta_r^2, \eta_\alpha^2\}$.

In order to linearize the non-linear system and measurement functions, a first-order approximation is used to compute the system dynamics matrix F and the measurement matrix H. The system functions are linearized around the current state estimate by taking the partial derivatives of the system function $f(\cdot)$ and the measurement function $h(\cdot)$ [2]. The Jacobian of the system dynamics matrix F and the measurement matrix H are derived as,

$$F = \left.\frac{\partial f(x)}{\partial x}\right|_{x=\hat{x}} \qquad H = \left.\frac{\partial h(x)}{\partial x}\right|_{x=\hat{x}} \tag{21}$$

To approximate the fundamental matrix ϕ_k, we need to use a Taylor-series expansion for e^{FT_s} [15] and implement the fundamental matrix ϕ_k as,

$$\phi_k = I + FT_s + \frac{F^2T_s^2}{2!} + \frac{F^3T_s^3}{3!} + \cdots \tag{22}$$

where T_s represents the sampling time period and I is an identity matrix. Generally, in order to simplify computation, only the first two terms of Eqn. (22) are chosen to approximate the fundamental matrix:

$$\phi_k \approx I + FT_s \tag{23}$$

The same equation set as the Kalman filter can be applied in our application. The subscript k to the Jacobians ϕ, W, H, and V indicates that they are different and need to be updated at each time step:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{24}$$
$$P_k^- = \phi_k P_{k-1}\phi_k^T + W_k Q_{k-1} W_k^T \tag{25}$$
$$K_k = P_k^- H_k^T (H_k P_k^- H_K^T + V_k R_k V_k^T)^{-1} \tag{26}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \tag{27}$$
$$P_k = (I - K_k H_k) P_k^- \tag{28}$$

The EKF is aimed at approximating the posterior PDF, $p(x_k|z_k)$ by linearizing the non-linear system functions. Many practical applications have obtained satisfactory results by using the EKF. However, with increasing level of system nonlinearity, the estimation error of the EKF is expected to increase. It is also very hard to tune the performance of the EKF. Moreover, the derivation of Jacobian matrices are difficult and sometimes even impossible to obtain [16].

3.2 Unscented Kalman Filtering

To mitigate the difficulties of the EKF implementation, Julier and Uhlmann(1997) developed the UKF to provide an alternative solution to the nonlinearity problems [7]. Like the EKF, the UKF is specific for nonlinear systems and it is a recursive MMSE (minimum mean square error) estimator. The UKF avoids using Jacobian matrices to linearize the non-linear system and measurement models. Instead, the UKF uses a method referred to as the unscented transformation (UT) to estimate the distribution of state variables. A set of sample points (sigma points) is deterministically selected by the UT. These sample points are then propagated through the true non-linear system functions and a set of transformed sample points is generated. Finally, the propagated mean and covariance of the transformed sample points are calculated to yield the posterior state estimate. The UKF is based on the assumption that it is easier to approximate a PDF than to linearize an arbitrary nonlinear function [7]. .

The first step of the UKF implementation is the state vector augmentation. A n-dimension state vector x needs to be restructured and augmented with q-term process noise. The dimension of the augmented state vector is $n^a = n + q$:

$$x_{k-1}^a = [x_{k-1}, w_{k-1}]^T \tag{29}$$

The process model can be rewritten as a function of x_{k-1}^a to calculate the *a priori* state estimate,

$$\hat{x}_k = f(x_{k-1}^a) \tag{30}$$

The augmented *a priori* state estimate and its covariance matrix are restructured as:

$$x_k^a = \begin{bmatrix} \hat{x}_k \\ 0_{q\times 1} \end{bmatrix} \qquad P_k^a = \begin{bmatrix} \hat{P}_k & P_{k_{xv}} \\ P_{k_{xv}} & Q_k \end{bmatrix} \tag{31}$$

where Q_k is the covariance matrix of the process noise and the input of the update stage with respect to the equations that we have previously given by Eqn.(26)–(28).

The algorithm of the unscented Kalman filtering is given as:

1) The augmented state vector x_k^a with the mean $\bar{x}^a$ and covariance P_k^a is approximated by $2n^a + 1$ weighted points given by

$$[\mathcal{X}_k, W_k] = sigmaPoints(x_k^a, P_k^a)$$

where $W_k \in \mathcal{R}^{2n^a+1}$ is a vector of weights, $\mathcal{X}_k = \{x_{i,k}\}_{\{i=1,\ldots,2n^a+1\}}$ defines $2n^a + 1$ sigma

points. The $2n^a + 1$ sigma points and the corresponding weights are generated by the function $sigmaPoints$ [8] to approximate the augmented state space x_k^a and the covariance P_k^a. The $sigmaPoints$ function is given as:

$$\begin{aligned} \mathcal{X}_{0,k-1} &= x_{k-1}^a & W_0 &= l/(n^a + l) \\ \mathcal{X}_{i,k-1} &= x_{k-1}^a + S_i & W_i &= 1/2(n^a + l) \\ \mathcal{X}_{i+n^a,k-1} &= x_{k-1}^a - S_i & W_{i+n^a} &= 1/2(n^a + l) \end{aligned}$$

where $i = 1, \ldots, n^a$; the term l is a fine tuner that can be used to reduce the overall prediction error. The term S_i is the ith column of the matrix square root of $(n^a + l)P_{k-1}^a$. The term S_i can be obtained by taking Cholesky factorization of $(n^a + l)P_{k-1}^a$:

$$\begin{aligned} A &= S_i S_i^T = (n^a + l)P_{k-1}^a \\ S_i &= cholesky(A) = \sqrt{(n^a + l)P_{k-1}^a} \end{aligned}$$

where $(n^a + l)P_{k-1}^a$ is a diagonal and upper triangular matrix. The matrix S_i is generated by Cholesky factorization and it is also an upper triangular matrix. The ith sigma point has its corresponding normalized weight W_i.

2) The set of sigma points is propagated through the process model to compute the transformed set of sigma points:

$$\mathcal{X}_{i,k} = f(\mathcal{X}_{i,k-1}, u_{k-1})$$

where $j = 1, \ldots, 2n^a + 1$

3) The *a priori* estimate is then computed by the weighted mean of the transformed sigma points and their corresponding weights:

$$\hat{x}_k^- = \sum_{i=1}^{2n^a+1} W_{i,k}(\mathcal{X}_k)_i$$

4) The last step of prediction stage is to compute the priori error covariance:

$$P_k^- = \sum_{i=1}^{2n^a+1} W_{i,k}[\hat{\mathcal{X}}_{i,k}^- - \hat{x}_k^-][\hat{\mathcal{X}}_{i,k}^- - \hat{x}_k^-]^T$$

5) In order to implement the update stage, we must compute the predicted observation of each sigma point $\mathcal{X}_{i,k}$ through the measurement function:

$$z_{i,k} = h(\mathcal{X}_{i,k}), i = 1, \cdots, 2n^a + 1$$

6) The priori expected measurement vector is computed by the weighted mean of the predicted observation for each sigma point:

$$\hat{z}_k^- = \sum_{i=1}^{2n^a+1} W_{i,k} z_{i,k}$$

7) Compute the error covariance matrices:

$$\begin{aligned} P_{\hat{z}_k\hat{z}_k} &= \sum_{i=1}^{2n^a+1} W_{i,k}(\hat{z}_{i,k} - \hat{z}_k^-)(\hat{z}_{i,k} - \hat{z}_k^-)^T + R \\ P_{\hat{x}_k\hat{z}_k} &= \sum_{i=1}^{2n^a+1} W_{i,k}(\hat{\mathcal{X}}_{i,k} - \hat{x}_k^-)(\hat{z}_{i,k} - \hat{z}_k^-)^T \end{aligned}$$

where:

• The term $P_{\hat{z}_k\hat{z}_k}$ is the innovation covariance and $P_{\hat{x}_k\hat{z}_k}$ specifies the cross correlation matrix.

• The term R is the covariance matrix of the measurement noise;

• The term $\hat{z}_{i,k}$ specifies the predicted measurement for each sigma point and $\hat{z}_k^-$ represents the priori expected measurement.

• The term $\hat{\mathcal{X}}_{i,k}$ is ith transformed sigma point and $\hat{x}_k^-$ specifies the priori state estimate.

8) Compute the Kalman gain:

$$K_k = P_{xz} P_{zz}^{-1}$$

9) Compute the *a posteriori* state estimate with the priori measurement vector $\hat{z}_k^-$:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{z}_k^-)$$

10) Finally, compute the *posteriori* estimate of the error covariance as follows:

$$P_k = P_k^- - K_k P_{\hat{z}_k\hat{z}_k} K_k^T$$

3.3 Particle Filtering and Bootstrap Algorithm

The EKF and UKF can only capture the first and second moments of state distribution of a Gaussian random variable. When the system model is highly nonlinear or the true distribution of state variable is non-Gaussian, both filters are prone to have large errors. The PF has the appealing convergence properties without being subject to any linearity or Gaussianity constraints. Particle filtering is a technique which implements the recursive Bayesian filter with Monte Carlo (MC) methods and sequential importance sampling (SIS) [17]. Unlike the UKF which uses a deterministic set of samples to approximate PDF, the PF uses point mass to represent probability densities. That is, the PF approximates

the posterior PDF by a set of random samples (particles) and their associated weights. The number of samples can be set sufficiently large to make the approximation as accurate as necessary [18]. As the number of samples approaches infinity, the performance of the posterior PDF approximation is equivalent to the optimal Bayesian estimate:

$$\lim_{N\to\infty} \hat{P}(x_k|z_k) \to P(x_k|z_k)$$

The Bayesian filter is aimed at constructing the posterior PDF of the state vector x_k given all available information. If $z_{1:k}$ denotes all the measurements up to the time step k, the posterior PDF at time step k can be expressed as $P(x_k|z_{1:k})$. The Bayesian recursive filter is composed of two phases: the prediction and the update phases. The prediction phase propagates the posterior PDF of the state vector of x_{k-1} forward to the priori state distribution x_k. Given $P(x_{k-1}|z_{k-1})$, the priori PDF of the state vector $P(x_k|z_{k-1})$ at $k > 0$ is obtained as [18]:

$$P(x_k|z_{1:k-1}) = \int P(x_k|x_{k-1})P(x_{k-1}|z_{1:k-1})dx_{n-1} \tag{32}$$

The update phase incorporates the new measurement z_k and updates the priori PDF to the posterior PDF of time step k:

$$P(x_k|z_{0:k}) = \frac{P(z_k|x_k)P(x_k|z_{1:k-1})}{P(z_k|z_{1:k-1})} \tag{33}$$

where the denominator of Eq. (33) $P(z_k|z_{1:k-1})$ can be extended as an expression of integrals:

$$P(z_k|z_{1:k-1}) = \int P(z_k|x_k)P(x_k|z_{1:k-1})dx_k \tag{34}$$

The integrals of Eq. (34) can be calculated by Monte Carlo methods. The posterior PDF $P(x_{0:k}|z_{1:k})$ is approximated by a set of N random samples $x^i_{0:t}$ with associated normalized weights $\mathcal{W}^i_k$ ($\sum_{i=1}^N \mathcal{W}^i_k = 1$). (Note: k is the index of the time step and i is the particle index). The weights $\mathcal{W}^i_k$ are referred to as the "probability mass" of their associated samples (particles) [18]. The posterior PDF is expressed in the discrete distribution $\sum_{i=1}^N \mathcal{W}^i_k \delta_{x^i_{0:k}}$.

Notice that it is often impossible to draw samples directly from the the posterior PDF $P(x_k|z_k)$ [17]. To overcome this difficulty, the importance sampling $q(x_k)$ is introduced. Suppose that the particle x^i_k is a sample drawn from a proposal distribution $q(x_{1:k}|z_{1:k})$ which is referred to as *importance weight*. We can define the weights as:

$$\mathcal{W}^i_k = \frac{P(x^i_{1:k}|z_{1:k})}{q(x^i_{1:k}|z_{1:k})} \tag{35}$$

In Bayesian processing, we need to sequentially estimate the posterior distribution $P(x_k|z_k)$ from $P(x_{k-1}|z_{k-1})$. The importance density $q(x_{1:k}|z_{1:k})$ also needs to be estimated sequentially.

The sequential representation of the *importance weight* [18] is given as:

$$\mathcal{W}^i_k = \mathcal{W}^i_{k-1}\frac{P(z_k|x^i_k)P(x^i_k|x^i_{k-1})}{q(x^i_k|x^i_{k-1}, z_{k-1})}, i = 1, \ldots, N, \tag{36}$$

A major problem of the SIS algorithms is the degeneracy of particles' weights. Since the variance of the importance weights increases over time, a few samples hold most of the importance weights, and other samples have zero importance weights in a long run [19]. This causes the algorithm to fail as there are not enough particles that can be used to approximate the posterior distribution. Moreover, a large portion of computational effort is wasted in propagating the particles with little weights and those particle would have a very small contribution to the representation of the posterior distribution.

An additional step known as resampling has to be introduced into the algorithm to resolve this degeneracy problem [20]. Resampling can be thought of as a procedure of eliminating particles with low probability (small weights) and replicating particles with high probability (large weights) [10]. Resampling involves mapping the weighted random measure $\{x^i_{0:k}, \mathcal{W}^i_{0:k}\}$ onto the equally weighted random measure $\{x^j_{0:k}, 1/N\}$. So after resampling, a set of N new particles with the same weights $\mathcal{W}^i_k = 1/N$ is generated to replace the empirical weighted distribution.

In our simulations and experiments, *Systematic Resampling*, the most widely used resampling algorithm, is applied in the update phase. The systematic resampling calculates the cumulative sum of the normalized weight $\mathcal{W}^i_c = \sum_{j=1}^i \mathcal{W}^j$ which is associated to each particle. A set of N ordered numbers with regular intervals of $1/N$ is defined and acts as an equally spaced "comb". The comb is offset by a random number drawn from a uniform distribution over $\mathcal{U}[1, N]$. The comb is then compared with the the cumulative sum of $\mathcal{W}^i_c$ to create a resampled set of numbers $u_{1:N} = \{u_j : \sum_{k=1}^{i-1} \mathcal{W}^k_n \leq u_j \leq \sum_{k=1}^{i} \mathcal{W}^k_n\}$ [9].

Fig.8 illustrates the systematic resampling scheme with an example of $N = 5$ samples. In

this example, the samples with the labels 1 and 4 in the original set are eliminated and the samples 3 and 5 are replicated once for each. Finally, the original set of samples $N = 1, \ldots, 5$ is translated to the resampled set which consists of labels of the original set as $2, 3, 3, 5, 5$.

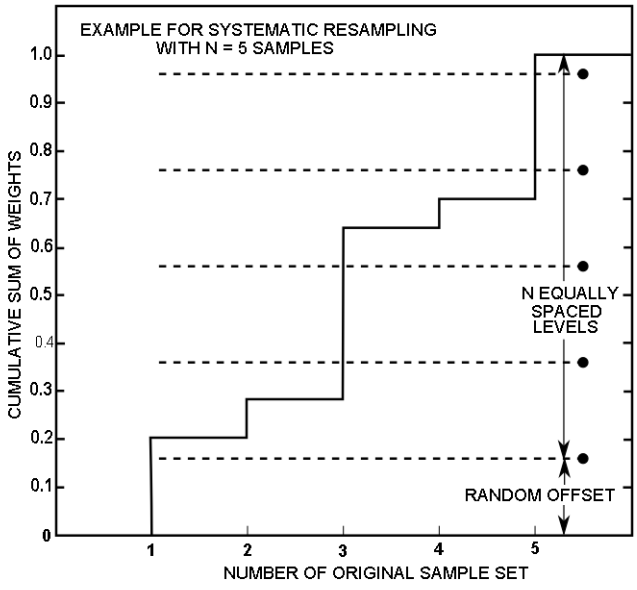

Fig. 8. Example of Systematic Resampling Scheme

The bootstrap algorithm used in the simulation was developed by Gordon, Salmond and Smith [9] in 1993. It is the first practical PF that was applied to tracking problems. The bootstrap PF and its variants are the most popular algorithms among all PF techniques due to its simplicity of implementation.

As mentioned previously, the PF degenerates due to the increasing variance of importance weights. A way to reduce the degeneracy is to find an importance distribution that minimizes the weight variance [18].

$$q(x_k|x_{k-1}, z_k) = P(x_k|x_{k-1}, z_k) \tag{37}$$

The proposal distribution can be decomposed by Bayes rule:

$$P(x_k|x_{k-1}, z_k) = \frac{P(x_{k-1}, z_k|x_k) \times P(x_k)}{P(x_{k-1}, z_k)} \tag{38}$$

The term $P(x_{k-1}, z_k|x_k)$ is expanded further as:

$$\begin{aligned} P(x_{k-1}, z_k|x_k) =& P(z_k|x_{k-1}, x_k) \times P(x_{k-1}|x_k) \\ =& P(z_k|x_{k-1}, x_k) \times \left[\frac{P(x_k|x_{k-1})P(x_{k-1})}{P(x_k)}\right] \end{aligned} \tag{39}$$

Substitute Eqn. (39) into Eqn.(38) and we have:

$$P(x_k|x_{k-1}, z_k) = \frac{P(z_k|x_{k-1}, x_k) \times P(x_k|x_{k-1}) \times P(x_{k-1})}{P(x_{k-1}, z_k)} \tag{40}$$

We can use Bayes' rule to expand the denominator of Eqn. (40) further:

$$P(x_{k-1}, z_k) = P(z_k|x_{k-1}) \times P(x_{k-1}) \tag{41}$$

Since the measurements are assumed to be independent of past states, thus we can write that:

$$P(z_k|x_{k-1}, x_k) = P(z_k|x_k) \tag{42}$$

Finally, the expression of the proposal distribution with the minimum variance is derived as:

$$P(x_k|x_{k-1}, z_k) = P(x_k|x_{k-1}) = q(x_k|x_{k-1}, z_k) \tag{43}$$

Leading to:

$$q(x_k|x_{k-1}, z_k) = P(x_k|x_{k-1}) \tag{44}$$

As shown in Eqn.(44), the bootstrap PF simply uses the prior distribution to calculate the importance sampling distribution. This means the most recent measurement data is not incorporated into the weight update.

The corresponding weight depends only on the likelihood $P(z_k|x_k)$ and its calculation is simplified from Eq. (36):

$$\begin{aligned} \mathcal{W}_k &= \mathcal{W}_{k-1} \times \frac{P(z_k|x_k)P(x_k|x_{k-1})}{P(x_k|x_{k-1})} \\ &= \mathcal{W}_{k-1} \times P(z_k|x_k) \end{aligned} \tag{45}$$

The simplicity of the implementation of the bootstrap filter is also its major shortcoming. Since the latest measurement does not contribute to the importance updating, this causes the variance of the weights to increase at each time step. In order to achieve convergence of the filter, resampling is required to be performed at each time step [18].

The bootstrap filter can be implemented as follows:

1) Initialization:
   a) Initialize the process noise covariance matrix as $Q = diag(\sigma_x^2, \sigma_y^2, \sigma_\theta^2)$ and measurement noise covariance matrix as $R = diag(\eta_r^2, \eta_\alpha^2)$.
   b) Initialize the *a priori* estimate error covariance matrix is P_x^-;
   c) Draw a set of initial samples from $x_0^i \sim P(x_0)$:

$$x_0^i = x_0 + \sqrt{(P_{x_0}^-)} * \mu, \quad \mu \sim \mathcal{N}(0, 1) \tag{46}$$

where $i = 1, \ldots, N$

d) Set $k = 1$.

2) Importance Sampling:

a) Draw a set of new particles $\hat{x}_k^{i(-)}$ to approximate the *a priori* importance density $q(x_k|x_{k-1}^i)$ as:

$$x_k^{i(-)} = f(x_{k-1}) + \sqrt{Q} \times \mu \times T \quad (47)$$

where $\mu \sim \mathcal{N}(0,1)$; T is the sample time period and $i = 1, \ldots, N$.
Note: as described previously, we use the prior distribution $x_k^{i(-)}$ only to calculate the importance sampling distribution in the bootstrap filter, thus we can set: $x_k^i = x_k^{i(-)}$.

b) Generate the likelihood: $P(Z_k|x_k^i)$ with the particle x_k^i and the latest measurement Z_k :
Calculate z_k^i through the measurement model $h(\cdot)$ with the newly generated particles x_k^i:

$$z_k^i = h(x_k^i); \quad (48)$$

Compute the likelihood of the current particle given the standard deviation of the measurement noise: η_r, η_α:

$$\mathcal{L}_k^i = \frac{1}{\eta_r\sqrt{2\pi}} \exp\left(-\frac{(z_{k,r} - z_{k,r}^i)^2}{2\sigma_r}\right) \times \frac{1}{\eta_\alpha\sqrt{2\pi}} \exp\left(-\frac{(z_{k,\alpha} - z_{k,r}^i)^2}{2\eta_\alpha}\right) \quad (49)$$

c) Assign each particle a weight, $\mathcal{W}_k^i$ by Eqn. (45) :

$$\hat{\mathcal{W}}_k^i = \mathcal{L}_k^i \times \mathcal{W}_{k-1}^i, i = 1, \ldots, N. \quad (50)$$

where $\mathcal{W}_{k-1}^i = 1/N$ after each resampling.

d) Normalize the importance weights:

$$\hat{\mathcal{W}}_k^i = \frac{\mathcal{W}_k^i}{\sum_{j=1}^N \mathcal{W}_k^j} \quad (51)$$

3) Resampling:
The bootstrap filtering algorithm requires to implement *Systematic Resampling* at each time step.
$[\{x_k^j, \mathcal{W}_k^j, i^j\}_j^N = 1] = \text{RESAMPLE}[\{x_k^i, \mathcal{W}_k^i\}_i^N = 1]$

a) At time step k, a set of particles with their associated weights is given: $\{x_k^i, \mathcal{W}_k^i\}$; $i = 1, \ldots, N$;

b) Initialize the CDF: $c_1 = 0$; Construct CDF: $c_i = c_{i-1} + \mathcal{W}_k^i, i = 2, \ldots, N$.

c) Sample $u_1 \sim \mathcal{U}(0,1)$ and start at bottom of CDF: $i^j = 1$.

d) Move along the CDF for $j = 2, \ldots, N$:
- Define uniform distributed variable: $\hat{u}_j = u_1 + \frac{j-1}{N}$;
- Determine the index i^j: $i = i + 1$ while $u_j > c_i$;
- Assign a new sample $x_k^j = x_k^i$ and it's weight: $\mathcal{W}_k^j = \frac{1}{N}$; assign parent $i^j = i$.

e) Finally, a new set of N particles and their weights $(1/N)$ are obtained as $\{x_k^j, \mathcal{W}_k^j\}$; for $j = 1, \ldots, N$.

4) Output:
Generate a new set as $\{\hat{x}_k^i, \hat{\mathcal{W}}_k^i\}$. The state is eventually updated by means of the particles as:

$$\hat{x}(k) = \frac{1}{N}\sum_{i=1}^N \hat{x}_k^i \quad (52)$$

5) Set $k = k + 1$ and go to step 2)

4 Numerical Analysis and Results

4.1 Simulation Results of Pose Estimation on A Three-Wheeled Mobile Robot

The EKF, UKF and the bootstrap PF are applied to perform pose estimation in the simulations. The performance and the computational time of each filter in terms of pose estimation are compared. The filters were implemented in MATLAB on a 2.4 GHz Intel Core 2 Quad processor. Each filter is applied to execute 50 runs of Monte Carlo simulation to perform the pose estimation. All filters were applied to estimate the system state $x_k = (x_k, y_k, \theta_k)^T$ recursively.

As described in Section 2.1, both the process model and the measurement model are non-linear. The mobile robot moves along a non-linear trajectory which is composed of straight lines and circles as shown in Fig.9. Three measurements including the range $r1$, $r2$ and the bearing α are available at each time step.

The bootstrap particle filter (BPF) is implemented with sample sizes of 500, 1000, 2500 respectively. The simulated mobile robot moves along a programmed path from 0s to 160s with the uniform

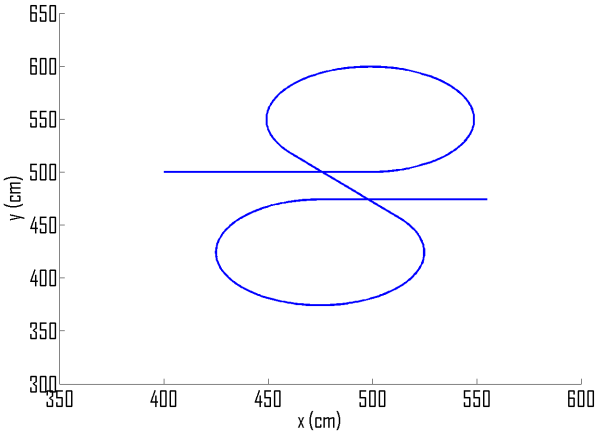

Fig. 9. The real path of Mobile Robot (solid line).

sampling period of $T = 0.2$s. All units are centimeter in length and radian in angle. The robot moves from its initial pose $(x, y, \theta)^T = (400, 500, 0)^T$.

The simulation executes 50 Monte Carlo trials using the EKF, the UKF and the bootstrap filter (sample size: 500, 1000, 2500) respectively. Each simulation trial consists of $N = 800$ epochs. The average RMSE (the root mean square difference between the true values and the state estimates) are computed by the following equation:

$$RMSE = \frac{1}{n}\sum_{j=1}^{n}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_j^i - \hat{x}_j^i)^2} \qquad (53)$$

where:

• x_j^i specifies the true simulated system state j for ith simulation. ($j = 1, \ldots, N$; $i = 1, \ldots, n$).

• $\hat{x}_j^i$ is the estimate of system state j for ith simulation.

• System state index: $j = 1, \ldots, N$,(In our case, $N = 800$).

• Simulation trial index: $i = 1, \ldots, n$, (In our case, $n = 50$).

The average RMS error and the variance of each filter are calculated with respect to the position and orientation. The comparison of position estimates is given in Table 1 and illustrated in Fig.10 and Fig.11. The comparison of heading estimate is given in Table 2 and illustrated in Fig. 12 and Fig.13. The computational time of each filter is given in Table 3 and illustrated in Fig. 14.

Notice that the UKF uses $2n^a + 1 = 19$ sigma points to perform pose estimation. Since the measurement model in the simulation is introduced in a nonlinear fashion, the augmented vector as shown in Eqn.(29) has to be expanded to include the observation terms [7]. Thus the dimension of the augmented state vector is $n^a = n + n_w + n_\eta = 3 + 3 + 3 = 9$. The augmentation introduces more sigma points which lead to more computational complexity of the UKF implementation. However, the measurement noise is processed by the algorithm and retains the same order of accuracy as the uncertainty in the state vector.

TABLE 1
Averaged RMS Position Error in Centimeters(50 Trials)

Algorithm	RMSE-Position	VAR
EKF	2.3864	0.0077
UKF(19 Sigma Points)	2.3748	0.0076
Bootstrap Filter(n=500)	1.0587	0.1293
Bootstrap Filter(n=1000)	0.8080	0.05390
Bootstrap Filter(n=2500)	0.4433	0.02223

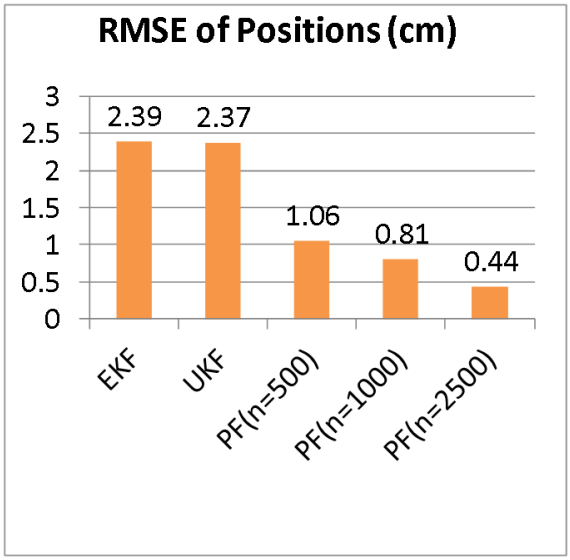

Fig. 10. Illustration of RMSE on Position Estimation

TABLE 2
Time-Averaged RMS Heading Error in Centimeters

Algorithm	RMSE-Heading	VAR
EKF	0.0164	1.5576E-7
UKF	0.0103	1.3545E-7
Bootstrap Filter(n=500)	0.0033425	3.4369E-7
Bootstrap Filter(n=1000)	0.0031204	1.8517E-7
Bootstrap Filter(n=2500)	0.0018395	1.0983E-7

Table 1 and 2 indicate that the EKF provides the worst performance among the three filters. The UKF has slightly better performance than the EKF

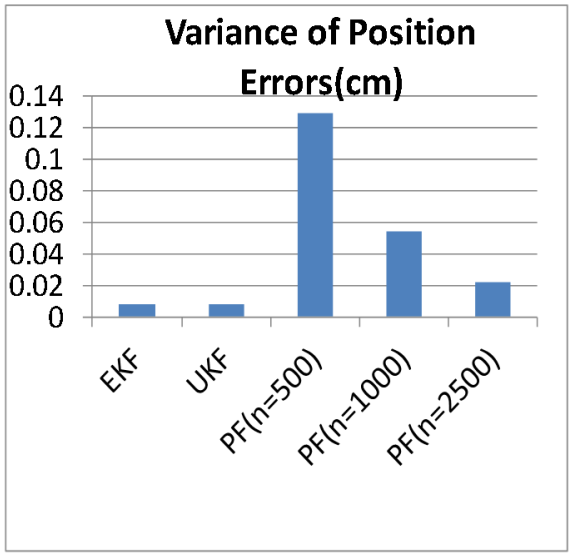

Fig. 11. Illustration of Variance of RMSE on Position Estimation

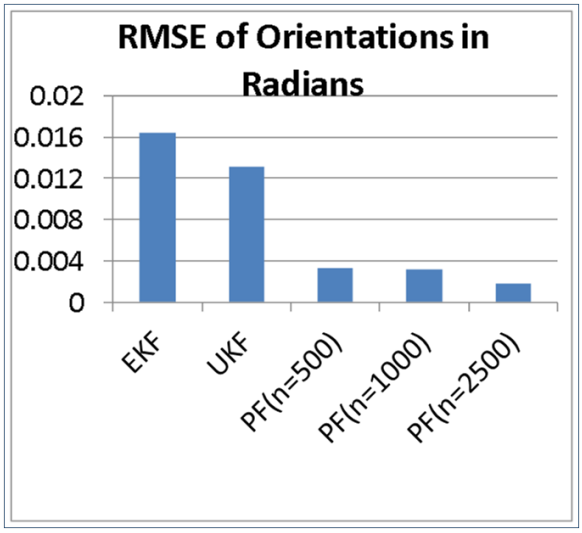

Fig. 12. Illustration of RMSE on Orientation Estimation

TABLE 3
Computational Time

Algorithm	Time(s)	Relative
EKF	1.5203	1.00
UKF	26.3848	17.36
Bootstrap PF (n=500)	866.23	569.78
Bootstrap PF (n=1000)	1738.13	1143.28
Bootstrap PF (n=2500)	4378.73	2880.18

on state estimate of position. However, the computational time of the UKF is more than 10 times than that of the EKF. The EKF is definitely a better choice

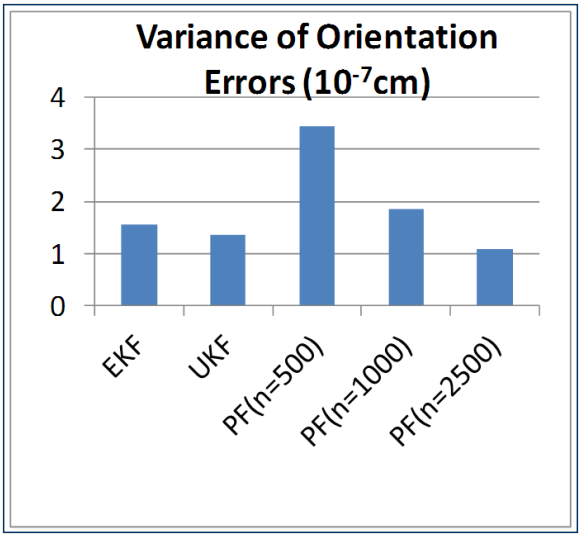

Fig. 13. Illustration of Variance on Orientation Estimation

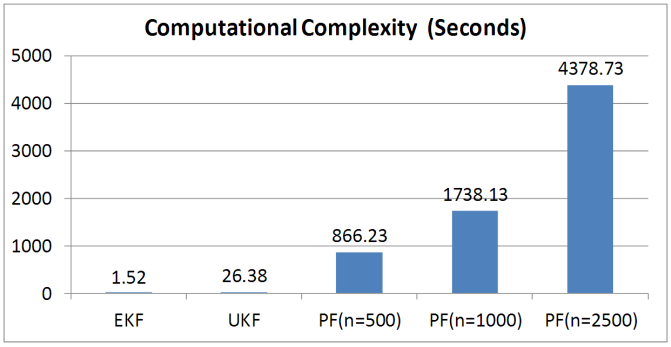

Fig. 14. A Comparison of Computational Time of the EKF, UKF and Bootstrap Filter with 500,1000,2500 particles

than the UKF on the robot pose estimation in our simulation.

It is also clearly noticeable in Fig.10 and Fig.12 that the bootstrap filter has a much smaller RMSE both on the position and orientation estimates in comparison to the EKF and UKF. Although the bootstrap filter leads to a much higher quality of state estimation, it takes around 860 seconds to perform the pose estimation (500 particles) for a trial with 160-sec navigation time. In order to implement the BPF in practice, it would be feasible to use parallel computation to implement a bootstrap PF in a high-level language such as C++.

Fig.17 illustrates the performance comparison of the bootstrap filter with $500, 1000, 2500$ samples. It shows that the average RMSE decreases as the number of samples increases. This is another advantage of the PF. Unlike the EKF and the UKF,

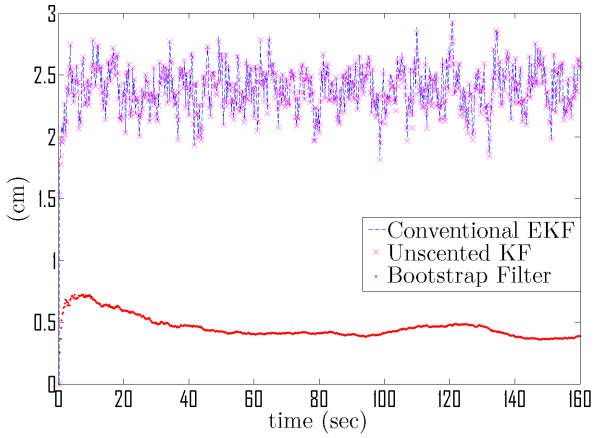

Fig. 15. A comparison of RMS errors of position: EKF,UKF,Bootstrap Filter(2500 samples)

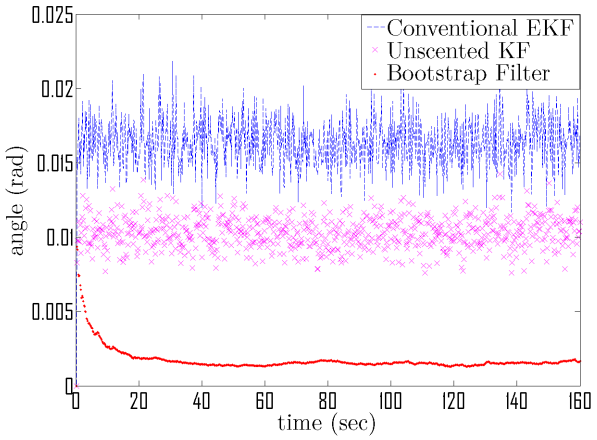

Fig. 16. A comparison of RMS errors of heading: EKF,UKF,Bootstrap Filter(2500 samples)

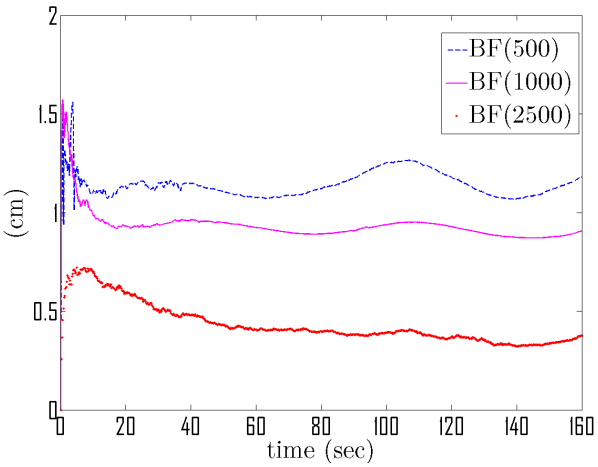

Fig. 17. A comparison of RMS errors for position of the EKF and Bootstrap Filter with 500,1000,2500 samples

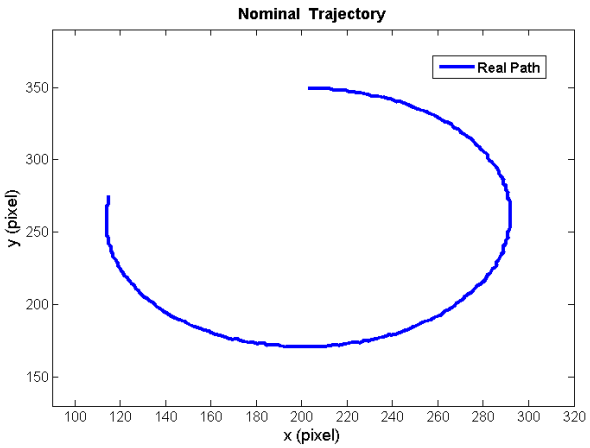

Fig. 18. The Camera Recorded Path of Mobile Robot in the Experiments(Solid line).

the performance of the PF can be easily tuned by adjusting the number of samples. We finally conclude that the bootstrap filter is the best filter if the computational cost is not our primary concern. However, if the computation cost is critical, the EKF is considered as the most efficient approach which also provides relatively good performance.

4.2 Experimental Results of Pose Estimation on A Differential Drive Mobile Robot

The non-linear filters including the EKF, UKF and the BPF are implemented to perform pose estimation on the basis of the system model and the measurement model in the experiments. The robots are programmed to move along a circular path. Two digital inputs which control the motors' speeds are recorded by an interrupt handler every 0.1 second. Simultaneously, the robot's trajectories are recorded by the camera systems. The data collected in the experiments are post-processed by the filters to perform pose estimation. The task of the non-linear filters is to estimate the system state $X_k = (x_k, y_k, \theta_k)^T$. Finally, the pose estimate of each filter are compared with each other and compared with the trajectory that is recorded by the camera system.

The mobile robot moves in a circular path in a clockwise direction as shown in Fig. 18. However, the robot's real paths often deviate from the programmed paths in the experiments due to the process noise. Fig.19 and Fig.20 illustrate the RMSE of the estimate of positions and headings with respect to each non-linear filter. As described earlier, the data is collected on-line by the robot control system

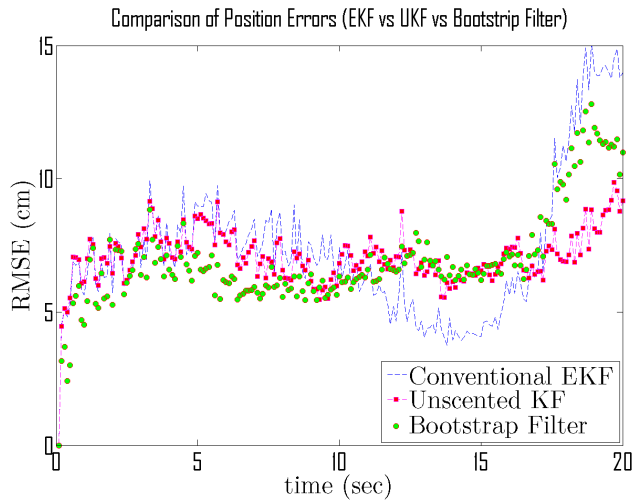

Fig. 19. A comparison of RMS errors of position: EKF,UKF,Bootstrap Filter(2500 samples)

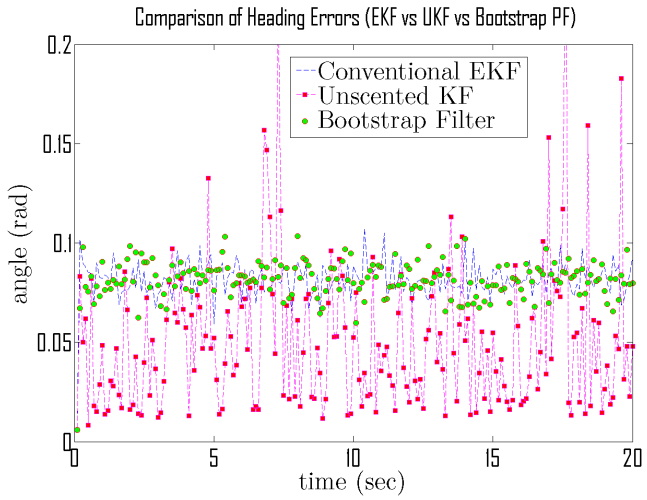

Fig. 20. A comparison of RMS errors of heading: EKF,UKF,Bootstrap Filter(2500 samples)

and the measurement system in the experiments. The collected data is then processed off-line by the non-linear filters to perform pose estimation. In order to compare the filter's performance on pose estimation, some statistical methods are employed. We performed 50 Monte Carlo trials with each non-linear filter to calculate pose estimates on the basis of the experimental data. The RMSE of the filters is averaged and then compared. The average RMSE (the root mean square difference between the recorded pose by the camera and the state estimates) are computed by Eqn.(53).

Th average RMSE and its variance of the robot position estimate are presented in Table 4 and illustrated in Fig.21. The RMSE and variance of

TABLE 4
Time-Averaged RMS Position Error in Centimeters

Algorithm	RMSE-Position	VAR
EKF	7.7979	1.2979
UKF	7.1039	0.8222
Bootstrap Filter(n=500)	7.1676	0.98542
Bootstrap Filter(n=1000)	6.8544	1.0388
Bootstrap Filter(n=2500)	6.8261	1.0168

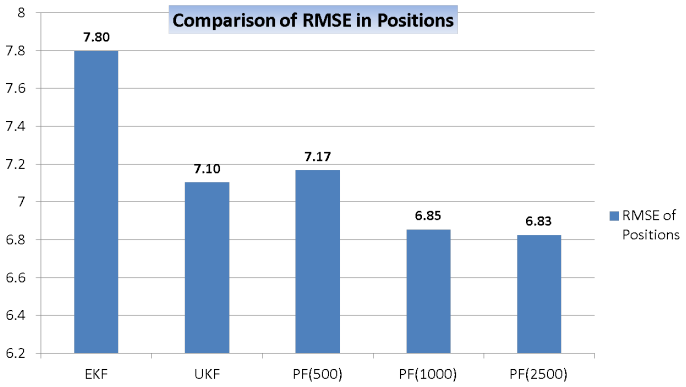

Fig. 21. Comparison of RMSE of position: the EKF, UKF, the Bootstrap Filter. (A illustration of data in Table 4)

TABLE 5
Time-Averaged RMS Heading Error in Centimeters

Algorithm	RMSE-Heading	VAR
EKF	0.081901	1.4480E-5
UKF	0.062322	1.2456E-5
Bootstrap Filter(n=500)	0.081878	2.2368E-5
Bootstrap Filter(n=1000)	0.081309	1.4738E-5
Bootstrap Filter(n=2500)	0.081030	8.5285E-6

TABLE 6
Computational Time

Algorithm	Time(s)	Relative
EKF	0.1857	1.00
UKF	3.3612	18.10
Bootstrap PF (n=500)	98.1565	528.58
Bootstrap PF (n=1000)	195.223	1051.3
Bootstrap PF (n=2500)	486.416	2619.4

the robot orientations are presented in Table 5. The computational time of each filter is presented in Table 6. It is clearly noticeable that the EKF's performance is not as good as the other two filters. However, it is still the most efficient algorithm. It takes only 0.18 seconds to complete the computa-

tion of the pose estimation. The UKF has a better performance than the EKF and the bootstrap filter (500 particles) on the state estimation. Although the UKF takes about 3.4 seconds (around 18 times more than the EKF) to perform the pose estimation, it is still much less than the 20 seconds of navigational time. Clearly, in our experiments, the UKF is a better choice than the EKF since the UKF can achieve a higher quality of pose estimation with relatively acceptable computational time.

The result of the simulations in Section 4.1 indicates that the bootstrap filter provides a much better performance over the EKF and UKF. However, based on the experimental results, the bootstrap filter can only provide a close or a slightly better performance over the UKF on pose estimation. The bootstrap filter (2500 particles) only outperforms by about 5% over the UKF in terms of the RMSE of positions. To perform the pose estimation for a trial of 20 seconds, the bootstrap filter takes around 100 seconds to complete the computation of 500 particles. Obviously, the computational time of the bootstrap filter is too high to perform on-line pose estimation in the experiments.

Table 4 also indicates that an increase in the number of particles can reduce the RMSE for the BPF. However, the change in performance is negligible as the number of particles increases from $N = 1000$ to 2500. Since the UKF has better performance than the EKF and much less computational time than the PF, we finally conclude that the UKF is the best filter to perform the state estimate in the experiments.

Unlike the simulation in Section 4.1, we determine the UKF as the best non-linear filter for the experiments. We notice that the level of non-linearity of the experimental models is lower than the non-linearity of the system models in our simulations. The camera system measures the state variables directly. That is, the measurement system of the experimental platform has a linear model. Whereas the measurement system has a non-linear model in the simulations. This is the major difference between the experiments and the simulations in terms of non-linearity.

Meanwhile, many factors contribute to producing the errors that have Gaussian and non-Gaussian distribution in the experiments. These errors include, but are not limit to the process noise, observational noise and model definition errors. Since it is difficult to evaluate some of these errors precisely such as the noise of the motors and the effect of the uneven wheels of the robot, the filters cannot achieve a better quality of pose estimation without evaluating of the effect of all these errors.

Based on the experimental result as shown in Table 4, the local linearization method of the EKF is not sufficient to approximate the posterior density of the experimental system. The UKF captures the first and second order moments of the state distribution of the Gaussian random variables. It appears to achieve a better representation of the experimental non-linear system over the EKF. The PF can perform the pose estimation without being subject to any linearity or Gaussianity constraints of the systems. However, its performance is only slightly better than the UKF. Furthermore, it is difficult to apply the PF to real time pose estimation due to its prohibitive computational time.

Finally, we can conclude that it is not necessary to perform the pose estimation with the PF. The UKF is the best choice to perform the pose estimation in the experiments.

5 Conclusion

This paper compares a brief outlook of three approaches to nonlinear filtering: the EKF, UKF and the BPF. To approximate the posterior PDF of state estimation, the EKF linearizes the system functions with a Taylor-series expansion. The implementation of the EKF is difficult and error prone since it has to take more effort to derive underlying Jacobian matrices. The UKF and the BPF approximate the PDF of state variable with a number of samples. The UKF uses a deterministic set of samples and generally has better quality on state estimation over the EKF. The BPF approximates the posterior PDF of state variables with a set of random samples.

In our simulations and experiments, the EKF is clearly the most efficient approach with the least computational load among all the three filters. The UKF needs more than 10 times the computational effort than the EKF. The EKF and the UKF are both practical filters to perform on-line robot pose estimation since their computational time is much less than the time of robot navigation. The computational time of the BPF implementation with 500 to 2500 particles is 500 to 3000 times more than that of the EKF. Normally, it is not practical to perform real time pose estimation with the PF unless some extra measures are applied such as the parallel computations.

The result of the simulations and experiments indicate that the UKF outperforms the EKF and

the PF has the best pose estimation accuracy. The PF has appealing performance on pose estimation on highly non-linear systems. However it generally cannot be applied on real time applications due to its prohibitive computational cost. For a system with a relatively low level of non-linearity, it is generally not necessary to perform pose estimation with the PF. The UKF and the EKF are both the practical choices to localize robots in real time applications. However, the PF is also a favourable choice if the system functions is highly non-linear.

The performance of the EKF and UKF is difficult to tune. This is another advantage of the PF. The performance of the PF can be easily tuned by adjusting the number of samples. If a parallel implementation is feasible, we can conclude that the particle-based filter is the best choice among the three nonlinear filtering approaches. However, the EKF and the UKF are certainly the options to be considered when the computational time is a critical concern such as in real time applications .

In the future, it would be interesting to investigate the adaptive particle filter (APF) which can automatically determine the number of needed particles over time. It would be interesting to compare the data from the three filters in this paper with the performance of the APF implementation which aims at finding a balance point between the computational load and the required accuracy.

REFERENCES

[1] S. F. Schmidt, "The kalman filter-its recognition and development for aerospace applications," *Journal of Guidance, Control, and Dynamics*, vol. 4, no. 1, pp. 4–7, 1981.

[2] P. Zarchan and H. Musoff, *Fundamentals of Kalman filtering: a practical approach*. AIAA, 2005, vol. 208, ch. 7.

[3] U. HENNING, J. Timmer, and J. Kurths, "Nonlinear dynamical system identification from uncertain and indirect measurements," *International Journal of Bifurcation and Chaos*, vol. 14, no. 06, pp. 1905–1933, 2004.

[4] J.-S. Gutmann and D. Fox, "An experimental comparison of localization methods continued," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1. IEEE, 2002, pp. 454–459.

[5] M. St-Pierre and D. Gingras, "Comparison between the unscented kalman filter and the extended kalman filter for the position estimation module of an integrated navigation information system," in *Intelligent Vehicles Symposium, 2004 IEEE*. IEEE, 2004, pp. 831–835.

[6] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 425–437, 2002.

[7] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *AeroSense'97*. International Society for Optics and Photonics, 1997, pp. 182–193.

[8] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[9] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," in *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2. IET, 1993, pp. 107–113.

[10] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: Particle filters for tracking applications*. Artech House Publishers, 2004, ch. 3.

[11] C. Samson, "Time-varying feedback stabilization of car-like wheeled mobile robots," *The International journal of robotics research*, vol. 12, no. 1, pp. 55–64, 1993.

[12] C. Samson, K. Ait-Abderrahim *et al.*, "Mobile robot control. part 1: Feedback control of nonholonomic wheeled cart in cartesian space," 1990.

[13] Arduino. (2012, 10) Arduino - introduction. [Online]. Available: http://arduino.cc/en/Guide/Introduction

[14] opencv dev team. (2013, 4) Opencv 2.5.4.0 documentation. [Online]. Available: http://docs.opencv.org/index.html

[15] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Journal of basic Engineering*, vol. 83, no. 3, pp. 95–108, 1961.

[16] S. J. Julier, "The scaled unscented transformation," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 6. IEEE, 2002, pp. 4555–4559.

[17] J. V. Candy, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Wiley-Interscience, 2011, vol. 54, ch. 3.

[18] ——, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Wiley-Interscience, 2011, vol. 54, ch. 7.

[19] A. Doucet, "On sequential monte carlo methods for bayesian filtering, dept. eng., univ. cambridge," UK, Tech. Rep, Tech. Rep., 1998.

[20] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, 2002.