2014 47th Hawaii International Conference on System Science

Tracking Targets under Uncertainty Natural Computing Approaches

Silja Meyer-Nieberg, Erik Kropat
Department of Computer Science
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany
Email: name.surname@unibw.de

Abstract

Tracking or more generally state estimation of dynamic systems are tasks that appear in many different contexts – for instance in surveillance with wireless sensor networks. Usually the state-evolution equations are assumed to be known excepting some parameters. In this case, particle filters and related approaches have been applied with great success. Very few attempts, however, have been made so far to address the problem of an unknown state equation. This paper presents approaches based on natural computing to solve this difficult and complex situation leading to a new kind of algorithms. Improvements to the original methods are introduced and investigated. The tracking quality is examined in simulations and compared to that of particle filters. The results show the performance of natural computing approaches are similar to that of particle filters for systems with known state-evolution equations. The new methods, however, can also be applied in situations with severe uncertainties.

Keywords

Tracking, evolution strategies, particle swarm optimization, particle filter, uncertainty, noise

1. Introduction

Tracking and localization tasks appear in various contexts ranging from civilian to military applications. Examples include the tracking of air planes in air traffic control and tracking tasks with wireless sensor networks. *Tracking* describes the problem of inferring the true state (e.g. the position) of a system by means of noisy sensor measurements and usually further information on the behavioral type of the system or target. Thus, estimating the unknown true state requires two models (see [2]): one describing the behavior, the other characterizing the measurements. If these are obtainable, they can be integrated into automated systems which take sensor measurements e.g. from a wireless sensor network and give an updated and more accurate estimate of the state or position. Usually particle filters and related methods are applied with great success for solving this task. This estimate can then be used to improve the common operational picture.

But what can be done if we cannot obtain a model for how the system behaves? Problems like these occur among others in ballistic target tracking or in the case of hand-held GPS-receivers [14]. In this case, particle filters cannot be applied since they require knowledge of the behavior type. This paper presents a new way to address this problem. The task can be compared to an uncertain search problem - that is, the method must cope with a dynamic and noisy situation. Therefore, one way to approach this problem is to make use of the inherent capabilities of evolutionary algorithms and other natural computing methods to track the uncertain position of a moving search point. This in turn leads to new challenges which will be described later. Several algorithms may be applicable – after adapting them to the task at hand. The aim is to provide first analyses of whether these methods can be used instead of particle filters in automated systems. Therefore, we focus on the algorithms and their operating principles.

We extend the results from [17], were we provided a first attempt at using natural computing for tracking. In the present paper, changes to the algorithms are proposed in order to cope with the specific task and explored. Additionally, a new method, particle swarm optimization is considered. Particle swarms have applied before in the area of tracking. However, the approaches considered versions developed for classical static optimization. Since tracking itself is dynamic, the application of variants introduced for dynamic optimization problems appears promising. Although we do not propose the use of natural computing as the main tracking operator in cases where the target behavior can be modelled quite well, it is nevertheless interesting to compare the performance of the natural computing approaches with that of particle filters. Therefore, the paper provides also a comparison with these methods.

The paper is organized as follows: It starts with a description of tracking and gives a sketch on particle filters and on so-called hybrid approaches. Afterwards, particle swarm optimization and modern evolution strategies are introduced. The next section gives the main ideas for the new algorithms which are investigated closer in the experimental section. Conclusions constitute the last part of the paper.

1.1. Tracking with Particle Filters

In tracking, variants of the general dynamic system

$$\begin{aligned} \mathbf{x}_{k+1} &= f_{k+1}(\mathbf{x}_k, \vec{\epsilon}_{k+1}) \\ \mathbf{z}_{k+1} &= h_{k+1}(\mathbf{x}_{k+1}, \vec{\omega}_{k+1}) \end{aligned} \quad (1)$$

are considered [2]. System (1) describes the evolution of the state variables $\mathbf{x}_k$ and the sequence of the measurements

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.150

$\mathbf{z}_k, k \geq 1$. The state variables are the "true" position of the target. Unfortunately, only noisy measurements $\mathbf{z}_k$ are available from which the positions have to be estimated. The random variables $\vec{\epsilon}_k$ and $\vec{\omega}_k$ denote measurement and process noise, respectively, and are assumed to be independent.

Tracking applications may focus on several tasks: Often the posterior density $p(\mathbf{x}_k|\mathbf{z}_{1;k})$, with $p(\mathbf{x}_k|\mathbf{z}_{1;k}) := p(\mathbf{x}_k|\mathbf{z}_1, \ldots, \mathbf{z}_k)$, is required which allows statistical estimates of interesting characteristics. In many cases, it is sufficient to derive some statistical moments - for instance the mean of the target position. The non-linear functions f_k, h_k, $k \geq 1$, are assumed to be known. Only a few approaches consider the possibility of unknown or uncertain parameters, e.g. [9], or allow model changes. The problem of estimating unknown parameters together with unknown state variables is known as *dual estimation* and can be approached in two ways: either by augmenting the state space with the uncertain parameters or by using two particle filters concurrently. Section 1.2 lists further methods from natural computing.

Following a Bayesian approach, the posterior density is obtained in two steps: a prediction and an update step. The *prediction step* obtains the density of the present target position given all previous measurements

$$p(\mathbf{x}_k|\mathbf{z}_{1;k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{z}_{1;k-1}) \mathrm{d}\mathbf{x}_{k-1}. \quad (2)$$

Once new measurements arrive, the update of the posterior reads

$$p(\mathbf{x}_k|\mathbf{z}_{1;k}) = \frac{p(\mathbf{x}_k|\mathbf{z}_{1;k-1}) p(\mathbf{z}_k|\mathbf{x}_k)}{p(\mathbf{z}_k|\mathbf{z}_{1;k-1})} \quad (3)$$

with

$$p(\mathbf{z}_k|\mathbf{z}_{1;k-1}) := \int p(\mathbf{z_k}|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{z}_{1;k-1}) \mathrm{d}\mathbf{x}_k. \quad (4)$$

Unfortunately, exact analytical solutions for (3) can only be obtained under quite restrictive assumptions. Otherwise approximations must be used. Among them is the well-known *particle filter* which is described shortly in the following. For a more detailed description, see e.g. [2, 6]. It can be shown that (3) can be approximated by

$$p(\mathbf{x}_k|\mathbf{z}_{1;k}) \approx \sum_{i=1}^{N_s} w_{i,k} \delta(\mathbf{x}_k - \mathbf{x}_{i,k}) \quad (5)$$

with N_s particles $\mathbf{x}_{i,k}$. The sum converges to the density for $N_s \to \infty$. The symbol $\delta(\mathbf{u})$ denotes the Dirac delta function and the $w_{i,k}$ are positive weights summing up to one.

The usage of a weighted density approximation is based on the following: Assume that there exists a density $p(\mathbf{x})$ from which it is difficult to sample but that another density $\tau(\mathbf{x})$ can be found that is easy to evaluate and fulfills $p(\mathbf{x}) \propto \tau(\mathbf{x})$. Still another density is required which allows to draw samples without difficulties. Using this *importance density* $q(\mathbf{x})$, $p(\mathbf{x})$ can be approximated by

$$p(\mathbf{x}) \approx \sum_{i=1}^{N_s} \frac{\tau(\mathbf{x}_i)}{q(\mathbf{x}_i)} \delta(\mathbf{x} - \mathbf{x}_i) \quad (6)$$

```
Algorithm 1:
A Generic Particle Filter (SIS)
PF^k :={(x_{i,k})_{i=1,...,N_s}, (w_{i,k})_{i=1,...,N_s}}
For i = 1,...,N_s
   Draw x_{i,k+1} ~ q(x|x_{i,k}, z_{k+1})
   Determine the weights
Determine the sum of the weights
For i = 1,...,N_s
   Normalize the weights
Test whether resampling is necessary
If TRUE
   RESAMPLE
PF^{k+1} :={(x_{i,k+1})_{i=1,...,N_s}, (w_{i,k+1})_{i=1,...,N_s}}
```

Figure 1. A generic particle filter. The description follows [2].

with the points $\mathbf{x}_i$ drawn from $q(\mathbf{x})$. In the case of particle filters, a general importance density would be given by $q(\mathbf{x}_{i,1;k}|\mathbf{z}_{i,1;k})$. This generic importance density leads using some assumptions (see [2]) finally to the general weight update equation

$$w_{i,k} = w_{i,k-1} \frac{p(\mathbf{z}_k|\mathbf{x}_{i,k}) p(\mathbf{x}_{i,k}|\mathbf{x}_{i,k-1})}{q(\mathbf{x}_{i,k}|\mathbf{x}_{i,k-1}, \mathbf{z}_k)} \quad (7)$$

with $q(\mathbf{x}_{i,k}|\mathbf{x}_{i,k-1}, \mathbf{z}_k)$ to be defined. Since the performance of the particle filter depends strongly on the importance density its good choice is critical. Often, the transition density $q(\mathbf{x}_k|\mathbf{x}_{i,k}, \mathbf{z}_k) = p(\mathbf{x}_k|\mathbf{x}_{i,k})$ is used, but this may lead to problems.

A common problem in particle filters is the so-called *degeneracy problem*, a kind of diversity loss of the population. After some time, typically only few particles will have larger weights – the majority of the particles does not contribute much to the density approximation. Therefore, the estimate is based on very few particles. According to [2] two general countermeasures exists: Choosing the proposal density very carefully helps to circumvent the problem whereas resampling the particles if the weight distribution becomes too uneven is another very common technique. Figure 1 shows a generic particle filter.

Several approaches exist which differ in the preconditions they make. Without resampling, the particle filter can be termed *sequential importance sampling* (SIS). A variant of SIS is for instance sampling importance resampling (SIR) which uses the importance density described above and resamples in every time step k. Further approaches include bootstrap filters [10] or convolution particle filters [9].

In this paper, we assume that the functions describing the evolution of the non-measurable state variables are unknown. Therefore, it is not possible to compute the transition density $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ explicitly. Only the measurements can be taken into account to derive and predict the target position. It should be noted that particle filters have strong similarities with

evolutionary algorithms (EAs) or particle swarm optimization, or more generally estimation of distribution algorithms (EDAs), which are used in optimization and search. The main difference is that EAs predefine a search distribution which is then adapted by selection and updated emulating principals from natural evolution. That is, they have their own state evolution equation which provided that it is possible to guide the search would allow the use of the approaches when the state equations are unknown. Before describing some of the approaches in more detail, we give a review on previous uses of natural computing in the area of tracking.

1.2. Hybrid Particle Filters and Dealing with Unknown Parameters

Natural computing is usually used together with particle filters resulting in so-called hybrid approaches that combine characteristics of several methods. Johansson and Lehmann [14] addressed the problem of dual estimation in which both the object position and its evolution have to be inferred. In their approach, they assumed that the model contained unknown parameters which had to be determined. Furthermore, ground truth was unavailable. To solve the problem, they combined a particle filter with an evolution strategy, a CMA-ES [12] for determining the unknown parameters. That is, a two-step process was applied with the ES completing the state equations and a bootstrap particle filter estimating the target position. The performance of the system was found to be good. For the optimization process, however, it was assumed that the target position can be extrapolated from previous positions which could potentially be problematic in other applications areas. Pengpai et al. [18] assumed that some parameters appearing in the state evolution and observation equation are unknown. They introduced a dual particle filter based on particle weight estimation by running two particle filters in sequence. However, the approach needs ground truth data for training. Yang [22] used a combination between particle filter, i.e., auxiliary particle filtering and particle swarm optimization [8] for dual estimation demonstrating the general ability of the new algorithm to track the state variables.

Hybrids between evolutionary algorithms and particle filters were not only considered for dual estimation. A series of papers by Uosaki et al., see e.g. [21], addresses combinations between evolution strategies and particle filters. To overcome the degeneracy problem they transferred the selection principle of evolution strategies and selected the particles with the largest weights. They obtained good results although not as good as those reached by the best particle filter.

1.3. Particle Swarm Optimization

Particle swarm optimization (PSO) is usually used for continuous search spaces. It is built after the swarming behavior of birds or fishes. In its simpliest form, the particles move through the search space by updating their velocity vector which indicates direction and extend of the movement. The update considers information of the swarm and of the search history. Usually, there are three main components

$$\mathbf{v}_k^{t+1} = \omega \mathbf{v}_k^t + c_1 \mathbf{r}_1 \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_{N_k}) + c_2 \mathbf{r}_2 \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_k) \qquad (8)$$

with $\cdot$ denoting componentwise multiplication. The first term $\omega \mathbf{v}_k^t$ is the old velocity which is included as a momentum term to safeguard against abrupt changes and to enable the swarm to leave the boundaries of the initial region. The second $\mathbf{x}_k - \hat{\mathbf{x}}_{N_k}$ is called the *social component*. The social component gives the particle the tendency to move towards the current best member $\mathbf{x}_{N_k}$ of the swarm or in its neighborhood.

This component is combined with stochastic influences $c_1 \mathbf{r}_1$ enforcing exploration. At this point, the swarm resembles a multi-point stochastic hillclimber. Ignoring the old velocities, all members of the swarm would move towards the current best solution. However, a particle also considers information from its own search history. The *cognitive component* $\mathbf{x}_k - \hat{\mathbf{x}}_k$ gives the particle the tendency to return to the best point it has found so far. Again, stochastic influences are present. The PSO described above is the original form of the so-called *global best PSO* since the best individual is determined using all swarm members. There are also types of PSO which consider local neighborhoods. Particle swarm optimization is quite efficient operating with swarm sizes of 10-30 individuals. Over the years, a lot of variants have been developed. The reader is referred to [8] for an overview. We will consider the global best PSO in this paper which uses the so-called *constriction coefficient*

$$\mathbf{v}_k^{t+1} = \chi\Big(\mathbf{v}_k^t + c_1 \mathbf{r}_1 \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_{N_k}) + c_2 \mathbf{r}_2 \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_k)\Big). \qquad (9)$$

It is easy to see that (8) and (9) can be transformed into each other by choosing the constants appropriately. It has been shown that PSO is able to track moving targets provided the change takes place in the space defined by the particles. There are specialized PSO approaches which aim at keeping up the diversity of the population enabling a better tracking capability. In this paper, one of these approaches, charged swarms, is implemented. In charged swarms, a part of the population is considered to carry a charge Q_k similar to the electric load in electron models. Since charged particles repell each other when their distance becomes too small, the swarm cannot collapse into a single point. The particle repulsion follows

$$\begin{aligned} \mathbf{a}_{km} &= \begin{cases} \frac{Q_k Q_m}{\|\mathbf{x}_k - \mathbf{x}_m\|^2} \frac{\mathbf{x}_k - \mathbf{x}_m}{\|\mathbf{x}_k - \mathbf{x}_m\|} & \text{if } d_{\min} \le \|\mathbf{x}_k - \mathbf{x}_m\| \le d_i \\ \frac{Q_k Q_m}{d_{\min}^2} \frac{\mathbf{x}_k - \mathbf{x}_m}{\|\mathbf{x}_k - \mathbf{x}_m\|} & \text{if } \|\mathbf{x}_k - \mathbf{x}_m\| \le d_{\min} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{a}_k &= \sum_{m \neq k} \mathbf{a}_{km}. \end{aligned} \qquad (10)$$

For each charged particle k the velocity update then changes to

$$\begin{aligned} \mathbf{v}_k^{t+1} &= \chi\Big(\mathbf{v}_k^t + c_1 \mathbf{r}_1 \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_{N_k}) + \\ &\qquad c_2 \mathbf{r}_2 \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_k) + \mathbf{a}_k\Big). \end{aligned} \qquad (11)$$

Since the charged swarm is already adapted to deal with dynamic optimization, we do not propose further changes to the algorithm.

1.4. Evolution Strategies

Evolutionary algorithms (EAs) [7] are population-based stochastic search and optimization algorithms. As direct search methods, they do require only weak preconditions on the function to be optimized and can be applied when only pointwise function evaluation is possible.

An evolutionary algorithm starts with an initial population. The individuals are either drawn randomly from the permissible search space or are initialized according to previous information. Each individual represents a candidate solution for the search or optimization problem. A subset of the population is selected as so-called parents in order to create new solutions. Creating new individuals encompasses usually two processes: recombination and mutation. *Recombination* combines traits from two or more parents, whereas *mutations* are random perturbations. After the offspring have been created, survivor selection is performed to determine the next parent population. Evolutionary algorithms differ in the ways they represent the solutions and implement selection, recombination, and mutation.

Evolution strategies (ESs) [19, 20, 4] are a variant of evolutionary algorithms that is predominantly applied in continuous search spaces. Mutation is the main search operator. This is in contrast to genetic algorithms which operate predominantly by recombination. Evolution strategies are commonly notated as $(\mu/\rho, \lambda)$-ESs. The parameter μ gives the size of the parent population, whereas λ stands for the size of the offspring population. If recombination is used, ρ parents enter the recombination result. Recombination is usually performed by calculating the weighted mean of the parents which is referred to as *intermediate recombination*. Other forms exist [4]. The result is then mutated – usually by adding a normally distributed random variable with zero mean and covariance matrix $\sigma^2\mathbf{C}$. Afterwards, the individuals are evaluated using the function to be optimized or a derived function which allows an easy ranking of the population.

Two main types of ESs can be distinguished: ES with "plus"-selection and ES with "comma"-selection. Evolution strategies with "plus"-selection choose the μ-best offspring and parents to form the next parent population, where ESs with "comma"-selection discard the old parent population completely. For continuous search spaces, "comma"-selection is usually recommended [4]. An important topic in evolution strategies is the continuous adaptation of the covariance matrix. Evolution strategies with ill-adapted parameters show slow convergence or are unable to find the optimal state at all. Therefore, methods for adapting the scale factor σ or the full covariance matrix have received a lot of attention (see [15]) – cumulating in evolution strategies with covariance matrix adaptation.

1.5. Updating the Covariance Matrix

First, the update of the covariance matrix is addressed. In evolution strategies two types exist: one used by the *covariance matrix adaptation evolution strategy* (CMA-ES) [13, 12] which considers past information from the search and an alternative used by the *covariance matrix self-adaptation evolution strategy* (CMSA-ES) [5] focuses on present information from the population.

The covariance matrix update of the CMA-ES is explained first. The CMA-ES uses weighted intermediate recombination, computing the weighted centroid of the μ best individuals of the population. This mean $\mathbf{m}^{(g)}$ is used for creating all offspring by adding a random vector drawn from a normal distribution with covariance matrix $(\sigma^{(g)})^2\mathbf{C}^{(g)}$, i.e., the actual covariance matrix consists of a general scaling factor (step-size) and the matrix denoting the directions. Following usual notation in evolution strategies this matrix $\mathbf{C}^{(g)}$ will be referred to as *covariance matrix* in the following.

The basis for the CMA update is the common estimate of the covariance matrix using the newly created population. Instead of considering the whole population for building the estimates, though, it introduces a bias towards good search regions by taking only the μ best individuals into account. Furthermore, it does not estimate the mean anew but uses the centroid $\mathbf{m}^{(g)}$. Following [13],

$$\mathbf{y}_{m:\lambda}^{(g+1)} := \frac{1}{\sigma^{(g)}}\left(\mathbf{x}_{m:\lambda}^{(g+1)} - \mathbf{m}^{(g)}\right) \; m = 1, \ldots, \mu$$

are determined with $\mathbf{x}_{m:\lambda}$ denoting the mth best off the λ particle according to the fitness ranking. The rank-μ update the obtains the covariance matrix as

$$\mathbf{C}_{\mu}^{(g+1)} \quad := \quad \sum_{m=1}^{\mu} w_m \mathbf{y}_{m:\lambda}^{(g+1)} (\mathbf{y}_{m:\lambda}^{(g+1)})^{\mathrm{T}} \qquad (12)$$

To derive reliable estimates larger population sizes are usually necessary which is detrimental with regard to the algorithm's speed. Therefore, past information, that is, past covariance matrizes are usually also considered

$$\mathbf{C}^{(g+1)} \quad := \quad (1 - c_\mu) C^{(g)} + c_\mu \mathbf{C}_{\mu}^{(g+1)} \qquad (13)$$

with parameter $0 \leq c_\mu \leq 1$ determining the effective time-horizon. In CMA-ES it has been found that an enhance of the general search direction in the covariance matrix is usual beneficial. For this, the concepts of the *evolutionary path* and the *rank-one-update* are introduced. As its name already suggests, an evolutionary path considers the path in the search space the population has taken so far. It is defined by the trajectory of the weighted means. With

$$\mathbf{v}^{(g+1)} := \frac{\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}}$$

the path is given as a combination of the old path and the new step

$$\mathbf{p}_c^{(g+1)} := (1 - c_c)\mathbf{p}_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}}\,\mathbf{v}^{(g+1)}. \quad (14)$$

For more details on the parameters see e.g. [12, 13]. The evolutionary path gives a general search direction that the ES has taken in the immediate past. In order to bias the covariance matrix accordingly, the rank-one-update is used

$$\mathbf{C}_1^{(g+1)} := \mathbf{p}_c^{(g+1)}(\mathbf{p}_c^{(g+1)})^{\mathrm{T}}. \quad (15)$$

A normal distribution with $\mathbf{C}_1^{(g+1)}$ leads towards a one-dimensional distribution on the line defined by $\mathbf{p}_c^{(g+1)}$. Together the components constitute the covariance update of the CMA-ES

$$\mathbf{C}^{(g+1)} := (1-c_1-c_\mu)\mathbf{C}^{(g)} + c_1\mathbf{C}_1^{(g+1)} + c_\mu\mathbf{C}_\mu^{(g+1)}. \quad (16)$$

The CMA-ES is one of the most powerful evolution strategies. However, as pointed out in [5], its performance does not scale very well with the population size. The CMSA-ES [5] updates the covariance matrix differently by considering

$$\mathbf{u}_{m:\lambda}^{(g+1)} := \frac{1}{\sigma_m^{(g+1)}}\left(\mathbf{x}_{m:\lambda}^{(g)} - \mathbf{x}_p^{(g)}\right) \quad (17)$$

with $\mathbf{x}_p^{(g)}$ the base vector of the mutation leading to $\mathbf{x}_{m:\lambda}^{(g+1)}$. Using (weighted) recombination, Eq. (17) equals the rank-μ update of the CMA-ES. The covariance update then reads

$$\mathbf{C}^{(g+1)} := (1-\frac{1}{c_\tau})\mathbf{C}^{(g)} + \frac{1}{c_\tau}\sum_{m=1}^{\mu} w_m \mathbf{u}_{m:\lambda}^{(g+1)}(\mathbf{u}_{m:\lambda}^{(g+1)})^{\mathrm{T}} \quad (18)$$

with the weights usually set to $w_m = 1/\mu$. See [5] for information on the free parameter c_τ.

1.6. Updating the Step-Size

The CMA-ES uses the so-called *cumulative step-size adaptation* (CSA) to adapt the scaling parameter (also called *step-size*, *mutation strength*, or *step-length*). To this end, the CSA [12] determines again an evolutionary path by summing up the movement of the population centers

$$\mathbf{p}_\sigma^{(g+1)} = (1-c_\sigma)\mathbf{p}_\sigma^{(g)} + \sqrt{c_\sigma(2-c_\sigma)\mu_{\mathrm{eff}}}(\mathbf{C}^{(g)})^{-\frac{1}{2}}\mathbf{v}^{(g+1)} \quad (19)$$

and eliminating the influence of the covariance matrix and the step length. For a detailed description of the parameters see [12]. The length of the path is important. If the path length is short, several movement of the centers counteract each other which is an indication that the step-size is too large and shall be reduced. If one the other hand, the ES takes several steps in approximately the same direction, progress and algorithm speed would be improved, if the ES could make larger changes. Therefore, long path lengths are seen as an indicator for a required increase of the step length. Ideally, the CSA should result in uncorrelated steps. After some calculations, see [12], it can be seen that the ideal situation is given by standard normally distributed steps, which leads to

$$\ln(\sigma^{(g+1)}) = \ln(\sigma^{(g)}) + \frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma^{(g+1)}\| - \mu_{\chi_n}}{\mu_{\chi_n}}\right) \quad (20)$$

as the CSA-rule. The parameter μ_{χ_n} in (20) stands for the mean of the χ-distribution with n degrees of freedom. If a random variable follows a χ_n^2 distribution, its square root is χ-distributed. The degrees of freedom coincide with the search space dimension. However, it can be shown that the original CSA encounter problems in large noise regimes resulting in a loss of step-size control and premature convergence. Therefore, uncertainty handling procedures and other safeguards are recommended.

An alternative approach for adapting the step-size is *self-adaptation* first introduced in [19] and developed further in [20]. It subjects the strategy parameters of the mutation to evolution. In other words, the scaling parameter or in its full form, the whole covariance matrix, undergoes recombination, mutation, and indirect selection processes. The working principle is based on an indirect stochastic linkage between good individuals and appropriate parameters: On average good parameters should lead to better offspring than too large or too small values or misleading directions. As stated, self-adaptation has been developed to adapt the whole covariance matrix. However, it is used nowadays mainly to adapt the step-size or a diagonal covariance matrix. In the case of the mutation strength, usually a log-normal distribution

$$\sigma_l^{(g)} = \sigma_{\mathrm{base}}\exp(\tau\mathcal{N}(0,1)) \quad (21)$$

is used for mutation. The parameter τ is called the *learning rate*. The variable σ_{base} is either the parental mutation strength or the result of recombination. For the step-size, it is possible to apply the same type of recombination as for the positions although different forms – for instance a multiplicative combination – could be used instead. The self-adaptation of the step-size is referred to as *σ-self-adaptation* (σSA) in the remainder of the paper. The newly created mutation strength is then used directly in the mutation of the offspring. If the offspring is sufficiently good, it is passed to the next generation. The baseline σ_{base} is either the mutation strength of the parent or if recombination is used the recombination result. Self-adaptation with recombination has been shown to be "robust" against noise [3] and is used in the CMSA-ES as update rule for the scaling factor. In [5] it was found that the CMSA-ES performs comparably to the CMA-ES for smaller populations but is less computational expensive for larger population sizes.

2. Natural Computing for Model-Free Tracking

This paper assumes that the state evolution equations are unknown and ground truth is unavailable. Therefore, tracking can only take the measurements and the observation equation into account

$$\mathbf{z}_k = h_k(\mathbf{x}_k, \vec{\omega}_k) \quad (22)$$

leading to the question how this information can be ultilized to guide the search. We assume that the density $p(\mathbf{z}_k|\mathbf{x})$ is obtainable at least approximately for $\mathbf{z}_k$ given, since this density is used as the fitness function

$$f_k(\mathbf{x}) \quad = \quad p(\mathbf{z}_k|\mathbf{x}) \tag{23}$$

for maximization or a derived measure with the same optimizer. Since $\mathbf{z}_k$ is the result of a noisy measurement, it appears as unfavourable to carry out the optimization of (23) to the end.

2.1. Coping with Noisy and Misleading Information

Information from the search so far is valuable to avoid moving towards false optima provided the true positional changes of the state variables are not too large and that no stark behavioral changes occur. It should be possible to recover the target movement to some extend by considering the search process of the algorithm. The maximizer of (23) provides an (inaccurate) estimate of the target position. The dynamic nature of the tracking task should be beneficial in itself. One the one hand, the reoccurring changes of the situation safeguard the population against converging towards a potential false optimizer and against loosing the population diversity to some extend. On the other hand, as the population tries to follow the target, the algorithm keeps statistics of the search process and is influenced by previous measurements. Therefore, it builds an implicit model of the target behavior. However, if the optimization run of the algorithm has sufficient time, it will concentrate on the optimizer of (23). It is possible to safeguard against this behavior by allowing only a few generations per se – even if no new measurement has arrived. This parameter must be tuned to the specific task. Another measure that may be taken is the introduction of artificial noise. As shown in [16], noise may sometimes be helpful in optimization since it may keep the algorithm from completing a subtask thus enabling the continued following of the overall goal. A similar reasoning can be applied here. It is not the goal to complete the optimization of (23) since the optimizer is misleading and its fulfillment would go hand in hand with a shrinkage of the population spread. Instead we want to use the fitness as a rough target to guide the search and maintain sufficient diversity. Artificial noise

$$\tilde{f}_k(\mathbf{x}) \quad = \quad p(\mathbf{z}_k|\mathbf{x}) + \epsilon \tag{24}$$

with $\epsilon \sim \sigma_\epsilon \mathcal{N}(0,1)$ may be helpful. This is the standard additive noise model in evolution strategies with σ_ϵ called the *noise strength*. The noise strength must be adapted carefully to the task. Whether articifial noise has a positive effect depends on the convergence behavior of the ES. The noise term will have an effect if the values of the true function are small. Therefore, we will consider a minimzation problem which is derived from (24) in order to assess the effects of artificial noise.

2.2. Evolutionary Particle Filters

The application of an ES to (23) is similar to using the density of a normal distribution as importance and transition density and substituting the resampling phase by rank selection based of the likelihood $p(\mathbf{z}|\mathbf{x})$. Therefore, in principle, evolution strategies and also particle swarms can be seen as special cases of particle filters. The following section introduces the *evolutionary particle filters* (EPFs) which are ESs that are adapted to tracking. This paper only considers EPFs with recombination notated as EPF_{rec}. While non-recombinative strategies are possible, the first investigations in [17] showed that intermediate recombination should be preferred.

2.2.1. Recombination. Recombination can be realized in several ways. The first decision is whether to use static or dynamic weights. In ES, static weights

$$(w_m)_{m=1,\ldots,\mu}, \text{ with } \sum_{i=m}^{\mu} w_m = 1$$

are common, either using equal weights $w_m = 1/\mu$ or following [12] in setting $w_m = \ln((\mu+1)/2) - \ln(m)$ and giving more weight to higher ranked individuals without considering the actual fitness differences. This is in contrast to typical particle filters which use a fitness dependent relative weight. However, as the investigations in [17] revealed using fitness dependent weights leads to instabilities. Therefore, constant weights are applied. Since the CMSA-ES usually operates with equal weights, we keep this setting for the EPF_{rec}-CMSA, whereas the EPS_{rec}-CMA will use unequal weights. Figure 2 sums up the main parts of the algorithms. Please note that the generation counter g is not related to the time index k.

2.2.2. Preserving the Diversity. New measurements can arrive during optimization leading to a dynamic optimization problem. In the case of evolution strategies, problems may appear if the magnitude of the change is very large which will lead to a longer adaptation times if the scaling factors have become too small. This will occur especially if measurements are taken infrequently. We propose two main approaches: a) no changes to the scaling factor in the assumption that in general convergence of the ES will take longer than the appearance of new measurements (this can be forced by limiting the search time of the ES) or b) transferring a concept stemming from particle swarm optimization. Aside from charged swarms, quantum particles have been introduced for dynamic optimization. This concept has not be carried over to evolution strategies until now although it appears helpful. Quantum particles are particles that do not move according to the usual rules. Instead they are created randomly around the population center. Usually, they are assumed to be normally distributed. If the population starts to converge into a single point, they keep up the diversity. Thus, they still enable searching up to a certain extend and serve as attractors for further development if they find better solutions than previously known. The concept has to be adapted in order to use them in evolution strategies.

If d denotes the index of a quantum individual, the individual is created as

$$\mathbf{x}_d \quad = \quad \mathbf{m} + \sigma_q \vec{\mathcal{N}}(\mathbf{0}, \mathbf{C}_q). \tag{25}$$

The generation index has been left out for clarity. The step-size and the covariance matrix are kept constant. In case, the resultant is selected for the next parent population, it is treated as a normal individual and takes part in the recombination processes and adaptation mechanisms. Selected quantum particles pull the mean of the population towards themselves enabling at least a gradual move out of the previous focus area. Additionally, there are influences on the step-size and covariance matrix. Since quantum particles increase the computational effort, they are only applied if the diversity provided by the usual mutation is low. As a measure we use the scaling parameter together with the squared root of the maximal eigenvalue of the covariance matrix since this can be seen as a representative of the effective step sizes.

2.2.3. Update of Covariance and Step-Size. In the case of the EPF_{rec}, the common ES-update rules for the covariance matrix are used. Also, for the scaling parameter, both approaches – the CSA-rule and the σSA-rule – appear applicable, although the time horizon for the evolutionary path in the CSA-rule is probably a decisive parameter. The σSA-rule is used with a small change by ultilizing the median instead of the mean in the recombination. It must be investigated whether using quantum particles together with the median type of recombination leads to significant effects. Since the median is robust against outliers, it may counteract the impact of the quantum particle. Therefore, the strategy will switch towards the mean whenever quantum particles are selected. Since all methods rely on past information, finding an appropriate generation time window is important. This paper considers the performance of systems with short generation time horizons which will require larger populations. The approaches are running systems, that is, they are started in the beginning of the measurements and incorporate new information as soon as a generation cycle is complete.

The following notation is used in the remainder of the paper. Let EPF_{rec}-CMSA stand for evolutionary particle filter with recombination using the update rules of the CMSA-ES, whereas EPF_{rec}-CMA applies the rules form the CMA-ES most notably the cumulative search path adaptation. The quantum variants will be denoted by QEPF_{rec}-CMSA and QEPF_{rec}-CMA. The first analyses will be carried out with the CMSA version. The PSO-based approach will be called particle swarm filter (PSF) in the remainder of the paper.

3. Experimental Setup

The algorithms have several parameters that can and should normally be tuned for a practical use. The experiments presented will apply mainly the default values of the PSO, CMA-ES, and CMSA-ES. An extensive research of the parameter dependence will be part of future work.

Algorithm 2:
Evolutionary Particle Filter with Recombination
g=0: Initialize $\sigma^{(0)} = \sigma_{\mu;k-1}$, $\mathbf{C}^{(0)} = \mathbf{C}_{k-1}$, $\mathbf{x}_m^{(0)} = \mathbf{m}_{k-1}$, $w_m^{(0)} = w_{m;k-1}$, $m = 1, \ldots, \mu$.
Calculate the weights

a) $$w_m^{(g+1)} = \frac{1}{\mu} \tag{26}$$

b) $$w_m^{(g+1)} = \ln\left(\frac{\mu+1}{2}\right) - \ln(m) \tag{27}$$

REPEAT
Compute the weighted mean

$$\mathbf{m}^{(g)} = \sum_{m=1}^{\mu} w_m^{(g)} \mathbf{x}_m^{(g)} \tag{28}$$

Create λ particles according to

$$\mathbf{x}_\lambda^l = \mathbf{m}^{(g)} + \sigma^{(g)} \mathcal{N}(\mathbf{0}, \mathbf{C}^{(g)}) \tag{29}$$

Determine the fitness

$$f(\mathbf{x}_\lambda^l) = p(\mathbf{z}_k | \mathbf{x}_\lambda^l) \tag{30}$$

Select the μ-best particles according to the best fitness values
Determine $\sigma^{(g+1)}$, $\mathbf{C}^{(g+1)}$
$g \to g+1$
UNTIL STOP
$\sigma_\mu = \sigma_\mu^{(g_{end})}$, $\mathbf{C}_k = \mathbf{C}^{(g_{end})}$, $\mathbf{m}_k = \mathbf{m}^{(g_{end})}$, $m = 1, \ldots, \mu$

Figure 2. Evolutionary particle filter (EPF_{rec}) using recombination. Note, in case of the CMA statistics of further parameter have to be kept.

The code for the experiments is based on the matlab-code `purecmaes.m` from N. Hansen [11] without improving the efficiency of the update rules. Therefore, system time will not be considered as a performance factor in itself. For the particle filter, we applied the code written by D. A. Alvarez [1].

3.1. Research Questions

The following questions are addressed in our first investigation of the algorithms using two dynamic models:

1) How well do the new variants perform compared to standard particle filter?
2) Does the introduction of quantum individuals improve the performance?
3) Is artificial noise helpful in dynamic tracking tasks?

3.2. Performance Measures

Following common practice in particle filter literature, we use the average root mean squared error (average RMSE)

$$\text{av. RMSE} := \frac{1}{M}\sum_{m=1}^{M}\sqrt{\frac{1}{T}\sum_{k=1}^{T}\|\hat{\mathbf{x}}(k,m)-\mathbf{x}(k,m)\|^2}$$

as performance measure with $\hat{\mathbf{x}}$ denoting the target position and $\mathbf{x}$ the output of the algorithms. The parameter T gives the measurement duration or the number of measurements for a experiment. We use M repeats.

3.3. Dynamic Systems

In this paper, two systems are considered to investigate the algorithms. A one-dimensional system inspired by [21], where only the step-size adaptation mechanism is explored and a three-dimensional system which requires the adaptation of the covariance matrix. In the case of the traditional particle filter, the systems will be used directly in the algorithm. The first system is given by

$$\begin{aligned} x_k &= \frac{x_{k-1}}{2}+\frac{25x_{k-1}}{1+x_{k-1}^2}+8\cos(1.2t)+\mathcal{N}(0,10) \\ z_k &= \operatorname{sign}(x_k)\frac{x_k^2}{20}+\mathcal{N}(0,1) \end{aligned} \tag{31}$$

with maximum of $p(z_i|x)$ directly discernible. No covariance matrix is necessary just step-size adaptation is required. The second system reads

$$\begin{aligned} \mathbf{x}_k &= \mathbf{F}\mathbf{x}_{k-1}+\mathbf{b}\cdot\vec{\mathcal{N}}(\mathbf{0},10\mathbf{I}) \text{ with} \\ \mathbf{F} &= \begin{pmatrix} 1 & T & \frac{T^2}{2} \\ 0 & 1 & T \\ 0 & 0 & 1 \end{pmatrix},\ \mathbf{b}=\begin{pmatrix} \frac{T^3}{6} \\ \frac{T^2}{2} \\ 1 \end{pmatrix} \\ \mathbf{z}_k &= \mathbf{x}_k+\vec{\mathcal{N}}(\mathbf{0},3\mathbf{I}) \end{aligned} \tag{32}$$

with $T = 0.25$.

3.4. Parameter Settings

We used the following settings for the algorithms. The EPF_{rec}-CMA applies the standard values taken from [11] with the exception of the offspring population size λ which was varied in the experiments. The size of the parent population was set to $\mu = \lambda/2$. The EPF_{rec}-CMSA followed the recommendation of using approximately $\mu = 0.25\lambda$. Regarding (18), the parameter was set to $c_\tau = 1/(\mu N)$. The learning rate, see (22) reads $\tau = 1/\sqrt{10}$. We are diverting from the usual recommendation of choosing $\tau \propto 1/\sqrt{N}$ since the dimension of the search space N is quite small and preliminary experiments revealed instabilities in the step-size adaptation.

We considered three different population sizes $\lambda = 50$, 100, and 200 together with four generation times $g_{end} =$ 2, 4, 6, and 20 resulting in twelve different combinations of parameter values. The PSF applied the following settings for the parameters: In the case of the update velocity (11), the parameters read $\chi = 0.953$, $r_1 = 2.05$, and $r_2 = 2.05$. For the determination of the particle repulsion $d_{\min} = 0.1$, $d_i = 0.5$, and $Q_k = -0.1$ were chosen. The number of charged particles was set 20% of the population. The EPFQ uses $\mu_q = \mu/2$ quantum individuals with a scaling factor of $\sigma_q = 1$ and covariance matrix $\mathbf{C}_q = \mathbf{I}$. The quantum individuals are used if the measure defined in Section 2.2.2 is smaller than 10 % of σ_q.

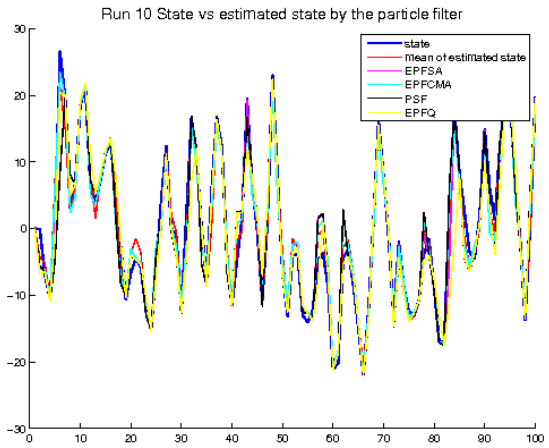

Figure 3. An exemplary run of the algorithms for System (31). The population sizes were set to 100 and the generation time was six. Not shown are the results for the QEPF-CMA.

We use just one realization of the dynamical system in order to make the results comparable. Otherwise, additional runs must be performed in order to eliminate the influences. This will be done in future research. The end time of the simulation was set to $T = 100$. We use 30 repeats for each parameter setting combination in case of System (31) and ten for (32). In order to assess the effects of noise, we set $\sigma_\epsilon^2 = 0.25$ and applied $f(x) = \|y - \operatorname{sign}(x)x^2/20\|^2$ for the minimization. The investigations were carried out for (31).

4. Results

This section discusses the results of the experiments. Figure 3 shows an exemplary run of the methods on the first dynamic system (31). As it can be seen, it is not easy to distinguish the results for the different approaches. A closer look at the corresponding box-plots, Fig. (4) reveals differences. Nearly all approaches have several outliers. In comparison to the spread of the outliers, the whiskers and the box are quite narrow. Especially the PSF and the QEPF-CMSA seem to be affected. The average RMSEs are reported in Tables 1 - 3. Since the QEPF-CMSA is always worse than the original EPF-CMSA, the results are not reported. Note, the EPF-CMA and the QEPF-CMA experienced numerical instabilities when

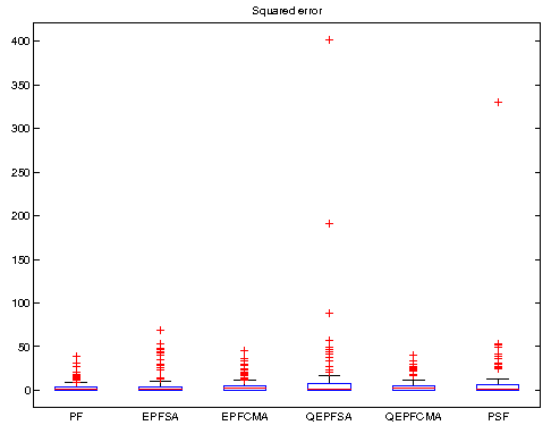

Figure 4. Exemplary box plots for the squared errors on (31). Shown are the corresponding results for Fig. 3, i.e., the results stem from the same run.

λ	g_{end}	PF	EPF-CMSA	EPF-CMA	QEPF	PSF
50	2	2.145	3.552	3.459	3.416	7.016
50	4	2.105	2.701	2.636	2.639	4.127
50	6	2.121	2.468	2.361	2.366	3.332
50	20	2.092	2.598	2.548	7642.005*	2.618
100	2	1.968	3.476	3.697	3.683	6.190
100	4	1.961	2.581	2.756	2.762	3.967
100	6	1.969	2.481	2.401	2.41	3.054
100	20	1.96	2.59	2.498	2.497	2.618
200	2	1.889	3.341	4.075	4.078	5.705
200	4	1.891	2.588	3.051	3.069	3.75
200	6	1.917	2.443	2.578	2.588	2.812
200	20	1.879	2.593	2.429	2.422	2.618

Table 1. Average RMSE for the first system (31). * Result caused by one extreme run. Otherwise the average would be around 2.5-2.6.

optimizing the density function for System (32). Therefore, a derived measure was used in the optimization. The EPF-CMSA and the PSF operate on the original function.

The average RMSEs for (31) are given by Table 1. As it could be expected, the particle filter has the lowest error since it can explicitly make use of the exact model. The other approaches are able to come close, however. They show a dependence on the number of allowed generations until new measurements arrive. Concerning the CMSA and the CMA versions, we find that if the population sizes are larger and the generation time is short, the EPF-CMSA leads to better results than the EPF-CMA. If the number of allowed generations is increased, the CMA variant appears preferable. According to the average RMSE, the quantum version does not have significant benefits on system (31). While the QEPF-CMA performs similar to the EPF-CMA and sometimes surpasses the results, no clear findings emerge. Further research appears necessary. In the case of the QEPF-CMSA, it is always worse

λ	g_{end}	PF	EPF-CMSA	EPF-CMA	QEPF	PSF
50	2	1.593	3.675	2.978	2.755	7.364
50	4	1.57	2.397	2.578	2.794	5.839
50	6	1.517	2.264	2.445	2.58	3.74
50	20	1.621	2.278	2.372	–*	71.06**
100	2	1.446	4.018	36.54	40.78	5.073
100	4	1.447	2.234	17.23	18.45	3.218
100	6	1.435	2.202	10.23	11.65	2.855
100	20	1.426	2.277	7.219	–*	19.147
200	2	1.404	4.618	443.9	442.6	4.859
200	4	1.402	2.237	569.3	547.7	2.672
200	6	1.4	2.201	664.8	828.1	2.578
200	20	1.3917	2.284	–*	–*	2.914

Table 2. Average RMSE for the second system (32). * All runs end with extreme errors. ** possibly caused by full convergence (possibly too few charged particles for $\lambda = 50$).

λ	g_{end}	PF	EPF-CMSA	EPF-CMA	QEPF	PSF
50	2	1.942	2.659	3.12	3.129	6.79
50	4	1.929	2.522	2.457	2.448	3.469
50	6	1.908	2.574	2.343	2.307	2.91
50	20	1.937	2.681	2.603	2.614	2.757
100	2	1.831	2.677	3.281	3.279	5.960
100	4	1.832	2.529	2.541	2.509	3.171
100	6	1.844	2.558	2.315	2.329	2.898
100	20	1.832	2.675	2.566	2.56	2.756
200	2	1.803	2.602	3.587	3.606	5.475
200	4	1.806	2.52	2.718	2.738	3.019
200	6	1.785	2.574	2.391	2.397	2.856
200	20	1.804	2.663	2.484	2.486	2.761

Table 3. Average RMSE for System (31) using noise.

than the EPF-CMSA. This may be due to the changed σ-update rule and will also be investigated closer. The charged PSF does not work well apparently when the generation time is too short. Interestingly, the number of particles does not have a strong impact on the performance. Thus, it would be possible and interesting to use the PSF with smaller populations but with longer generation times. This will be investigated in more detail in further work.

System (32) reveals that the CMA-variants may experience difficulties for larger population sizes. In comparison, the EPF-CMSA works quite well. Since the CMA versions apply a different covariance matrix update, a closer look at the mechanism is required. The PSF should probably operate with more quantum particles than in the setting used.

For System (31), noise appears to have a beneficial effect for some constellations as Table 3 shows. While this is promising, further investigations are necessary since we had to switch to a minimization task and therefore had to change the function to be optimized.

5. Conclusions

This paper discusses the application and adaptation of natural computing approaches for tracking. Tracking represents a state estimation problem for systems where only

noisy measurements are available and has many important applications. Usually particle filters and related approaches are applied. They require a valid model of the system behavior and provide then an improved estimate of the true state or position. This model, however, may not always be obtainable. Natural computing methods may offer a way out. They are similar to particle filters in many aspects but use their own intrinsic behavioral model which is adapted according to feedback from the search process. This ability is the motivation for applying natural computing to tracking. This paper analyses two dynamic systems. The traditional particle filter serves as a comparison reference using the behavior models directly, while the natural computing methods considered only the measurements. The natural computing methods perform slightly worse than the particle filter with respect to estimating the true position, but the quality of the results is at least comparable. Since the natural computing methods are applicable where particle filters fail, this is a very promising result motivating further research. Tracking can be interpreted as a dynamical noisy search task. Potential challenges for the algorithm include a diversity loss and therefore the risk of being unable to follow the moving state. Additionally, a convergence towards a false optimizer must not occur. Therefore, the natural computing methods were adapted. The first change introduces the concept of quantum particles in order to preserve the population diversity. The other explores the use of artifical noise in order to keep the algorithms away from deceptive optima.

References

[1] Alvarez, D. A. (2005). Particle filter tutorial. http://www.mathworks.com/matlabcentral/fileexchange/35468-particle-filter-tutorial, accessed on 13.06.13.

[2] Arulampalam, M., S. Maskell, N. Gordon, and T. Clapp (2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing 50*(2), 174–188.

[3] Beyer, H.-G. and S. Meyer-Nieberg (2006). Self-adaptation of evolution strategies under noisy fitness evaluations. *Genetic Programming and Evolvable Machines 7*(4), 295–328.

[4] Beyer, H.-G. and H.-P. Schwefel (2002). Evolution strategies: A comprehensive introduction. *Natural Computing 1*(1), 3–52.

[5] Beyer, H.-G. and B. Sendhoff (2008). Covariance matrix adaptation revisited - the CMSA evolution strategy -. In G. Rudolph et al. (Eds.), *PPSN*, Volume 5199 of *Lecture Notes in Computer Science*, pp. 123–132. Springer.

[6] Doucet, A. and A. M. Johansen (2011). A tutorial on particle filtering and smoothing: Fifteen years later. In D. Crisan and B. Rozovsky (Eds.), *Oxford Handbook of Nonlinear Filtering*. Oxford University Press.

[7] Eiben, A. E. and J. E. Smith (2003). *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin: Springer.

[8] Engelbrecht, A. P. (2005). *Fundamentals of Computational Swarm Intelligence*. Wiley.

[9] Gauchi, J.-P. and J.-P. Vila (2013). Nonparametric particle filtering approaches for identification and inference in nonlinear state-space dynamic systems. *Statistics and Computing 23*(4), 523–533.

[10] Gordon, N., D. Salmond, and C. Ewing (1995). Bayesian state estimation for tracking and guidance using the bootstrap filter. *Journal of Guidance, Control, and Dynamics 18*(6), 1434–1443.

[11] Hansen, N. (2005). CMA-ES source code. http://www.lri.fr/∼hansen/cmaes_inmatlab.html, accessed on 26.07.13.

[12] Hansen, N. (2006). The CMA evolution strategy: A comparing review. In J. Lozano et al. (Eds.), *Towards a new evolutionary computation. Advances in estimation of distribution algorithms*, pp. 75–102. Springer.

[13] Hansen, N. and A. Ostermeier (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*(2), 159–195.

[14] Johansson, A. and E. Lehmann (2009, aug.). Evolutionary optimization of dynamics models in sequential monte carlo target tracking. *Evolutionary Computation, IEEE Transactions on 13*(4), 879 –894.

[15] Meyer-Nieberg, S. and H.-G. Beyer (2007). Self-adaptation in evolutionary algorithms. In F. Lobo, C. Lima, and Z. Michalewicz (Eds.), *Parameter Setting in Evolutionary Algorithms*, pp. 47–76. Heidelberg: Springer Verlag.

[16] Meyer-Nieberg, S. and H.-G. Beyer (2008). Why noise may be good: Additive noise on the sharp ridge. In M. Keijzer et al. (Eds.), *GECCO 2008: Proceedings of the 10th annual conference on genetic and evolutionary computation*, pp. 511–518. ACM.

[17] Meyer-Nieberg, S., E. Kropat, and S. Pickl (2013). Evolutionary particle filters. In B. Vitoriano and F. Valente (Eds.), *Proceedings of ICORES 2013*, pp. 96–102. SCITEPRESS.

[18] Pengpai, F., S. Li-Fen, W. Bing, and W. Wei (2009). Particle filter-weight estimation and dual particle filter. In *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*, pp. 1–4.

[19] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog Verlag.

[20] Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Chichester: Wiley.

[21] Uosaki, K. and T. Hatanaka (2005). Evolution strategies based gaussian sum particle filter for nonlinear state estimation. In *The 2005 IEEE Congress on Evolutionary Computation*, Volume 3, pp. 2365 – 2371.

[22] Yang, X. (2012). Particle swarm optimisation particle filtering for dual estimation. *Signal Processing, IET 6*(2), 114 –121.