Accelerating Neuroevolutionary Methods Using a Kalman Filter

Yohannes Kassahun
Robotics Group
University of Bremen
Robert-Hooke-Str. 5, D-28359
Bremen, Germany
kassahun@informatik.uni-bremen.de

Jose de Gea
Robotics Group
University of Bremen
Robert-Hooke-Str. 5, D-28359
Bremen, Germany
jdegea@informatik.uni-bremen.de

Mark Edgington
Robotics Group
University of Bremen
Robert-Hooke-Str. 5, D-28359
Bremen, Germany
edgimar@informatik.uni-bremen.de

Jan Hendrik Metzen
Robotics Lab
German Research Center for Artificial Intelligence (DFKI)
Robert-Hooke-Str. 5, D-28359
Bremen, Germany
jhm@informatik.uni-bremen.de

Frank Kirchner
University of Bremen
German Research Center for Artificial Intelligence (DFKI)
Robert-Hooke-Str. 5, D-28359
Bremen, Germany
frank.kirchner@informatik.uni-bremen.de

ABSTRACT

In recent years, neuroevolutionary methods have shown great promise in solving learning tasks, especially in domains that are stochastic, partially observable, and noisy. In this paper, we show how the Kalman filter can be exploited (1) to efficiently find an optimal solution (i. e. reducing the number of evaluations needed to find the solution), (2) to find solutions that are robust against noise, and (3) to recover or reconstruct missing state variables, traditionally known as state estimation in control engineering community. Our algorithm has been tested on the *double pole balancing without velocities* benchmark, and has achieved significantly better results on this benchmark than the published results of other algorithms to date.

Track-name: Genetics-Based Machine Learning and Learning Classifier Systems

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

General Terms

Algorithms

Keywords

Neuroevolution, Kalman Filter

1. INTRODUCTION

Stochastic partially-observable problems have always been a challenging domain for machine-learning algorithms. The fundamental reason for this is that such problems place limits on an agent's ability to fully perceive the states of the environment, and in so doing, limit the information upon which an agent can base its decisions. Such problems cannot be solved using learning algorithms that assume the underlying process is a Markov Decision Process (MDP) since complete observability is necessary for this assumption to be valid [9]. Various methods have been employed to reduce the effects of this limited information. One example involves basing decisions on a history of recent observations and actions [14].

In neuroevolutionary methods, policies are represented by neural networks, and the effect of limited (incomplete) information is often dealt with by using recurrent connections within a network to recover unobserved state variables from the state variables that have been observed. A major drawback of such networks is the difficulty in training them, which means that they require a larger number of evaluations before a solution is found.

In this paper, we propose an alternative to using recurrent neural networks in neuroevolution for overcoming the effects of stochastic partially-observable domains. This new method reduces the number of evaluations needed to get a solution. Instead of recovering unknown or noisy state variables with recurrent neural networks, we use the POMDP agent [9] shown in Figure 1. This agent is composed of two parts: the state estimator and the policy. The state estimator estimates the current state b based on the previous state, the last control (action) u, and the current observed (measured) state z. The policy π maps the state to a control (action) u. The technique presented in this paper combines the efficiency of the Kalman filter [11] (state estimator) with the agility of feed-forward neural networks (policy), dramatically improving the speed at which policies can be searched.

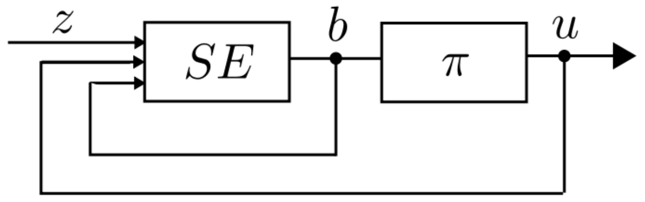


Figure 1: A POMDP agent. ***SE*** **is a state estimator and π is a policy represented by a feed-forward neural network.**

This paper is organized as follows: first, a brief review of works in the area of neuroevolution that have been tested on the double pole balancing velocities benchmark will be presented, followed by a short introduction to algorithms used in our approach. After this, we provide a detailed description of our approach, and the results that have been obtained with it. Finally, we offer some conclusions and possible further research directions related to the presented approach.

2. REVIEW OF WORKS TESTED ON DOUBLE POLE BALANCING WITHOUT VELOCITIES BENCHMARK

In this section we will give a review of neuroevolutionary methods that are tested on the double pole balancing velocities benchmark. One can divide the methods into two major categories. The methods in the first category evolve only the weights of the neural networks for a given topology, while methods in the second category evolve both the topology and weights of neural networks.

2.1 Methods that Evolve the Weights of a Neural Network

Wieland has investigated the evolutionary optimization of fully connected recurrent neural networks on different pole balancing problems [21]. He encoded the weights of the neural networks using eight bits and used a genetic algorithm for optimization. The structure of the recurrent neural network was manually determined. Saravanan and Fogel used Evolutionary Programming (EP) for parameter optimization of feed-forward neural networks on the double pole balancing benchmark [16]. The structure of the neural network, that is the number of hidden units, are determined a priori. Symbiotic Adaptive Neuroevolution (SANE) [15] is an evolutionary system that evolves a population of neurons instead of a population of networks. Fully connected hidden layers of networks are formed by a combination of neurons selected randomly from a population of neurons. A neuron individual receives an average fitness value from the networks in which it takes part. The Enforced Subpopulations (ESP) method [3] is based on SANE, but it specializes neurons to specific tasks. Each non-input unit of the neural network is assigned to a separate subpopulation and a neuron is recombined with the members of its own subpopulation. Unlike SANE, the networks formed by ESP consist of a representative from each evolving specialization. This allows ESP to evolve recurrent networks, because a neuron's behavior in a recurrent network critically depends on the neurons to which it is connected. A specialized evolutionary strategy called CMA-ES [7] (see Section 3.1) was used to evolve a fixed-topology neural network for solving the double pole balancing benchmark [8]. This method can evolve both forward and recurrent fixed-topology neural networks for the pole balancing problems. The Cooperative Synapse NeuroEvolution (CoSyNE) method [4] uses cooperative coevolution to construct neural networks. For a given user-specified network architecture, a population consisting of n subpopulations is created, where n is the number of weights in the network to be evolved. In addition to the standard genetic operators, the algorithm permutes each subpopulation to increase diversity in the population.

2.2 Methods that Evolve Both Weights and Topologies of Neural Networks

Gruau's Cellular Encoding (CE) method is a language for local graph transformations that controls the division of cells which grow into an artificial neural network [5]. Through cell division, one cell, the *parent cell*, is replaced by two cells, the *child cells*. During division, a cell must specify how the two child cells will be linked. The genetic representations in CE are compact because genes can be reused multiple times during the development of the network. This saves space in the genome because not every connection and node need to be explicitly specified in the genome. Defining a crossover operator for CE is still difficult, and it is not easy to analyze how crossover affects the subfunctions in CE since they are not represented explicitly. The NeuroEvolution of Augmenting Topologies (NEAT) [19] evolves both the structure and weights of neural networks using crossover and mutation operators. It starts with networks of minimal structure and increases their complexity along the evolution path. Every node and connection of a phenotype (neural network) is encoded by a corresponding genotype. The algorithm keeps track of the historical origin of every gene that is introduced through structural mutation. This history is used by a specially designed crossover operator in order to match up genomes encoding different network topologies, and to create a new structure that combines both the common and the differing parts of two parent structures. Structural discoveries of the evolutionary process are protected by niching (speciation). The speciation in NEAT is achieved by explicit fitness sharing, where organisms in the same species share the fitness of their niche. NEAT does not use self-adaptation of mutation step-sizes. Each connection weight is perturbed with a fixed probability by adding a floating-point number chosen from a uniform distribution of positive and negative values. The Evolutionary Acquisition of Neural Topologies (EANT [13]) uses a novel encoding of neural networks [12] that is suitable for evolving networks in both direct and indirect encoding scenarios. For evolving networks in a direct encoding scenario, a nature inspired meta-level evolutionary process is used, where the exploration of structures takes place over a larger timescale, and the exploitation of existing structures (i. e. the optimization of its weights) is done on a smaller timescale. Analog Genetic Encoding (AGE) [2] is an implicit genetic encoding which is based on genetic regulatory networks in biological systems. The method is primarily developed to evolve analog electrical circuits, but it can also be used to evolve neural networks.

3. BRIEF INTRODUCTION TO THE ALGORITHMS USED

In this section we will give a very brief introduction to the algorithms used in our approach: CMA-ES and the Kalman filter.

3.1 CMA-ES

Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [7] is an advanced form of evolution strategy [17], which can perform efficient optimization even for small population sizes. Each individual is represented by an $n-$dimensional real valued solution vector. The solutions are altered by recombination and mutation. Mutation is realized by adding a normally distributed random vector with zero mean, where the covariance matrix of this distribution is itself adapted during evolution to improve the search strategy. CMA-ES uses important concepts like *derandomization* and *cumulation*. Derandomization is a deterministic way of altering the mutation distribution such that the probability of reproducing steps in the search space that lead to better individuals is increased. A sigma value represents the standard deviation of the mutation distribution. The extent to which an evolution has converged is indicated by this sigma value (smaller values indicate greater convergence). Moreover, the algorithm detects correlations between object variables (i. e. variables in the vector to be optimized), and is invariant under orthogonal transformations of the search space. Correlations between object variables are detected by analyzing the search path of a population over several past generations. These correlations are stored in the covariance matrix and guide the future search path in a promising direction. This principle is known as cumulation.

3.2 Kalman Filter

The Kalman filter [11] estimates the state of a linear dynamical system that is perturbed by a gaussian noise. Formally, the filter addresses a general problem of estimating the true state $x \in \mathcal{R}^n$ of a discrete linear time system governed by

$$x_k = A_k x_{k-1} + B_k u_{k-1} + w_{k-1}, \tag{1}$$

where A_k is an $n \times n$ state transition matrix, B_k is an $n \times m$ control input model matrix, $u_k \in \mathcal{R}^m$ is the control vector, and w_k is the process noise which is assumed to be drawn from a zero-mean multivariate normal distribution with covariance matrix Q_k of size $n \times n$. The measurement (observation) $z_k \in \mathcal{R}^l$ of the true state is modelled by

$$z_k = C_k x_k + v_k, \tag{2}$$

where C_k is an $l \times n$ matrix representing the measurement model and v_k is the measurement noise which is again assumed to be drawn from a zero mean multivariate normal distribution with covariance matrix R_k of size $l \times l$.

The Kalman filter recursively estimates the current state based on the current measurement and the estimate from the previous state. The filter has basically two distinct phases: *predict* and *update*. Let $P_{k|k-1}$ and $\hat{x}_{k|k-1}$ be the *a priori* estimate of the error covariance matrix and the true state at timestep k, respectively, and $P_{k|k}$ and $\hat{x}_{k|k}$ be the *a posteriori* estimate of the error covariance matrix and the true state at timestep k, respectively. The filter starts with initial estimates for the true state $\hat{x}_{k-1|k-1}$ and the error covariance matrix $P_{k-1|k-1}$, and then repeatedly executes its *predict* and *update* phase routines. The equations of Kalman filter in the two phases are given in Table 1. Refer to [20] for a more detailed introduction to the Kalman filter.

Table 1: Equations of the *predict* and *update* phases of the Kalman filter.

Predict Phase

Predict state:
$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1|k-1} + B_k u_{k-1}$

Predict error covariance:
$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_{k-1}$

Update Phase

Kalman gain:
$K_k = P_{k|k-1} C_k^T (C_k P_{k|k-1} C_k^T + R_k)^{-1}$

Update state estimate:
$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - C_k \hat{x}_{k|k-1})$

Update error covariance:
$P_{k|k} = (I - K_k C_k) P_{k|k-1}$

4. ACCELERATING NEUROEVOLUTIONARY METHODS

In this section, a new approach for accelerating neuroevolutionary methods is presented, which is based on the POMDP agent shown in Figure 1. Specifically, the agent that is developed with this approach takes the form of a neural network augmented with a Kalman-filter based predictor. First, the augmented neural network will be discussed, followed by a description of the Kalman filter implementation that was used. We then finish with an explanation of how such augmented neural networks can be optimized.

4.1 Description of the Augmented Neural Network

The main idea behind our approach of accelerating neuroevolutionary methods is to augment an evolving neural network with a predictor that can estimate the next state based on the current partially-observable state (which is possibly corrupted by noise). The predictor we use is composed of n Kalman filters $\{KF_1, KF_2, \ldots, KF_n\}$, one for each of the n sensory readings, as shown in Figure 2. The outputs of these Kalman filters are connected to a feed-forward neural network NN, whose outputs control the plant. The whole controller is a non-linear function given by

$$u_j = f(w_1, \ldots, w_n, \sigma_{w1}, \sigma_{v1}, \ldots, \sigma_{wn}, \sigma_{vn}, z_1, \ldots, z_n), \tag{3}$$

where u_j ($j \in [1, m]$) is one of the outputs of the neural network, $w_1, \ldots, w_n$ are the weights of the neural network, σ_{wi}, σ_{vi} are the standard deviation values of the the measurement and process noise, respectively, of the Kalman filter KF_i, and $z_1, \ldots, z_n$ are the measured values (inputs to the controller).

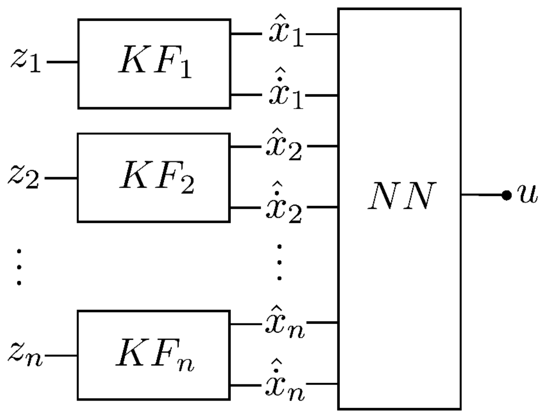


Figure 2: The augmented neural network: the Kalman filter KF_i is used to estimate the sensor value $\hat{x}_i$ and the missing value $\hat{\dot{x}}_i$ from the measured (observed) value z_i. The quantity u_j represents the control signal that is sent to the plant to be controlled. NN is a feed-forward neural network.

4.2 The Steady-state Kalman Filter with Constant Velocity Model

The Kalman filter used in our implementation is a particular type of the general Kalman filter in which a constant velocity model is assumed. The constant velocity model is usually used in tracking applications [10, 1] and is also known as an $\alpha\beta$ filter. Because the double pole benchmark task forces the system to work in a limited region where velocity does not change dramatically, we are able to assume a constant velocity model. The steady-state version of the Kalman filter is used in cases where the time required to compute the algorithm is an important constraint. For a given system, one can let the Kalman filter run for several cycles and record the Kalman gains K in steady state. These will be constant, so the computation can easily be sped up by always using these constants instead of updating K each cycle (which requires a matrix inversion computation). The equations that describe the steady-state Kalman filter are:

$$\begin{aligned}\hat{x}_{k|k-1} &= A \cdot \hat{x}_{k-1|k-1} && (4)\\ \tilde{y}_k &= z_k - C \cdot \hat{x}_{k|k-1} && (5)\\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K \cdot \tilde{y}_k && (6)\end{aligned}$$

where $\hat{x}_{k|k-1}$ represents the estimate of x at time k given observations up to and including time $k-1$. z_k is the measurement at time k, A is the state transition matrix, C is the output array and K is the steady-state Kalman gain. Expression (5) computes the innovation factor that allows the predictions to be updated after new measurements have been obtained. Given the assumption of a constant velocity model, the filter will choose two weighting coefficients (α and β) that will weight the differences between predictions and new measurements when updating the current prediction to find a new estimate. To better illuminate this, consider the classical tracking equations for the $\alpha\beta$ filter:

$$\begin{aligned}x_p(k) &= x_s(k-1) + v_s(k-1)T && (7)\\ v_p(k) &= v_s(k-1) && (8)\\ x_s(k) &= x_p(k) + \alpha(z_k - x_p(k)) && (9)\\ v_s(k) &= v_s(k-1) + (\beta/T)(z_k - x_p(k)) && (10)\end{aligned}$$

where $x_p(k)$ and $v_p(k)$ are the predicted position and velocity at time k, $x_s(k)$ and $v_s(k)$ are the smoothed position and velocity at time k, T is the sampling time, and α and β are the weighting coefficients. After calculating $x_p(k)$ and $v_p(k)$ (eqns. 7 and 8), the calculation of the smoothed parameters only requires the proper selection of values for α and β. The optimal values for α and β have been derived by Kalata [10], and depend on the assumed variance of both measurement and process noises (σ_v and σ_w):

$$\begin{aligned}\gamma &= \frac{T^2 \cdot \sigma_v}{\sigma_w} && (11)\\ r &= \frac{4 + \gamma - \sqrt{8 \cdot \gamma + \gamma^2}}{4} && (12)\\ \alpha &= 1 - r^2 && (13)\\ \beta &= 2 \cdot (1 - r)^2 && (14)\\ K &= \begin{bmatrix} \alpha \\ \beta/T \end{bmatrix} && (15)\end{aligned}$$

The state transition matrix A is initialized the with a constant velocity model:

$$A = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \qquad (16)$$

and the output array C is

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad (17)$$

where the '1' in the first column indicates that we have measurements from the position, and the '0' in the second column indicates that we have no information about the velocity.

In our experiment, we use 3 Kalman filters, one for each of the measured variables (cart position, pole 1 position, and pole 2 position).

4.3 Optimizing the Augmented Neural Network

The topology of the neural network can be determined manually, or automatically using a neuroevolutionary algorithm that has this capability. In both cases, the neuroevolutionary algorithm must optimize both the weights of the neural network, and the process and measurement noise parameters of the Kalman filters. For example, it is possible to optimize the weights of the network and the process and measurement noise parameters of the Kalman filters in the exploitation phase of the EANT algorithm [13], [18] using CMA-ES.

Noise-free Partially Observable Domains

In partially observable noise-free domains (e. g. the standard double pole balancing without velocities benchmark) there are some state variables that are not observable. The purpose of the Kalman filter in this case is simply to estimate the missing variables. Therefore, we can set both the measurement noise σ_{vi} and the process noise σ_{wi} of each Kalman filter KF_i to very small values and optimize only the weights of the neural network.

Noisy Partially Observable Domains

Partially observable domains which contain noise are the most general case, since virtually all real-world problems are noisy and partially observable. In this case the Kalman filter not only has to predict the missing state variables, but must also filter the noise. It is therefore necessary to optimize the measurement noise σ_{vi} and process noise σ_{wi} of each Kalman filter KF_i, as well as the weights of the neural network. After optimization, a solution will be found that is robust against noise.

Practically, it is possible to determine the variance of the measurement noise σ_{vi}^2 by taking off-line sample measurements. The determination of the variance of the process noise σ_{wi}^2, however, is generally more difficult since one is not usually able to directly observe the process being estimated [20].

5. EXPERIMENTS AND RESULTS

In this section we show the results obtained using our approach to solve the double pole balancing without velocities benchmark. First, a brief introduction to both the benchmark problem is given, as well as an explanation of the fitness function from the literature used to evaluate the performance of the agent (controller). Afterwards, our POMDP agent's performance will be evaluated in two different scenarios: (1) in a noise-free scenario in order to compare it with the current state-of-the-art results, and (2) in a noisy environment to show its robustness against noise in comparison with other methods. All experimental results in this paper are obtained using topologies that are manually determined and optimized using the CMA-ES algorithm.

5.1 The Double Pole Balancing Problem Without Velocities Benchmark

The pole balancing system has one or more poles hinged to a wheeled cart on a finite length track. The movement of the cart and the poles are constrained within a 2-dimensional plane. The objective is to balance the poles indefinitely by applying a force to the cart at regular time intervals, such that the cart stays within the track boundaries. An attempt to balance the poles fails if either (1) the angle from vertical of any pole exceeds a certain threshold, or (2) the cart leaves the track boundaries.

In the double pole balancing without velocities benchmark, the controller observes only x, θ_1, and θ_2, but *not* $\dot{x}$, $\dot{\theta}_1$, and $\dot{\theta}_2$. A fitness function introduced by Gruau et al. is used in connection with this benchmark [6]. The fitness function is the weighted sum of two separate fitness measurements $f = 0.1f_1 + 0.9f_2$ taken over 1000 timesteps.

$$f_1 = t/1000$$

$$f_2 = \begin{cases} 0 & \text{if} \quad t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}\left(|x_i|+|\dot{x}_i|+|\dot{\theta}_{1,i}|+|\dot{\theta}_{2,i}|\right)} & \text{otherwise,} \end{cases} \tag{18}$$

where t is the number of time steps the pole is balanced starting from a fixed initial position. In the initial position, all states are set to zero except $\theta_1 = 4.5°$. The angle of the poles from the vertical must be in the range $[-36°, 36°]$. The defined fitness function favors controllers that can keep the poles near the equilibrium point and minimize the amount of oscillation. The first fitness measure f_1 rewards successful balancing, while the second measure f_2 penalizes oscillations. The evolution of the neural controllers is stopped when a champion of a generation passes two tests. First, it has to balance the poles for 10^5 timesteps starting from the 4.5° initialization. Second, it has to balance the poles for 1000 steps starting from at least 200 out of 625 different initial starting states. Each start state is chosen by giving each state variable $(x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ one of the values 0.05, 0.25, 0.5, 0.75, 0.95,0, 0, scaled to the range of each input variable. The ranges of the input variables are ± 2.16 m for x, ± 1.35 m/s for $\dot{x}$, $\pm 3.6°$ for θ_1, and $\pm 8.6°$ for $\dot{\theta}_1$. The number of successful balances is a measure of the generalization performance of the best solution.

The fitness function f is not directly proportional to the performance of an individual in the two tests. Therefore, at the end of a given generation, the controller with the highest fitness is *not necessarily* the controller which performs better on the two tests. It is possible that there could be another controller which was assigned a lower fitness, but has better performance on the two tests. This forces neuroevolutionary methods to be more explorative, and therefore results in a larger number of evaluations needed to solve the benchmark.

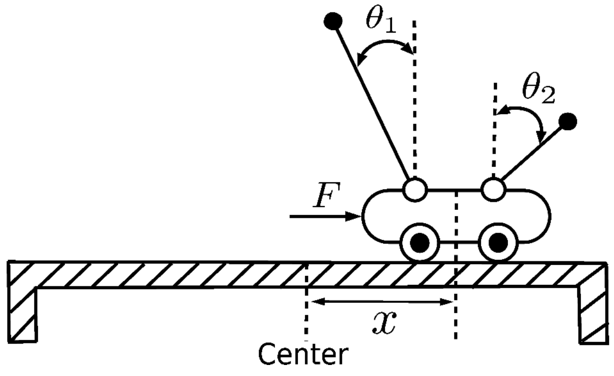


Figure 3: The double pole balancing problem. The poles must be balanced simultaneously by applying a continuous force F to the cart. The parameters x, θ_1 and θ_2 are the offset of the cart from the center of the track, and the angles from the vertical of the long and short pole, respectively.

5.2 Performance Evaluation

The first performance evaluation experiment was performed in a noise-free environment so that our approach could be compared with the results in the literature. In this scenario, the velocities of the cart and poles are not measurable and the task of the Kalman filters is to estimate those non-observable variables. The measurement noise parameter of each filter was set to $\sigma_{vi} = 10^{-7}$, and the process noise parameter to $\sigma_{wi} = 10^{-3}$. With these values the Kalman filter assumes that there is very little noise. In other words, it assumes that both measurements and state transitions will be reliable. The first estimation is initialized with

Table 2: Results for the double pole balancing without velocities benchmark. Average over 50 simulations. Generalization refers to the average number of successful balances from 625 different initial positions.

	Evaluations	Generalization
CE	840000	300
SANE	451612	-
ESP	169466	289
NEAT	33184	286
AGE	25065	317
EANT	15762	262
CMA-ES	6061	250
CoSyNE	3416	-
CMA-ES + Kalman	302	334

$\hat{x}_i(0) = x_i(0)$, where $x_i(0)$ takes on the initial value for the position of the cart, pole 1's position, or pole 2's position, depending on which Kalman filter is considered. These values are initialized as described for the benchmark problem: the cart and pole 2 initial values are set to zero, and the initial value for pole 1 is set to 4.5°. In this configuration, the Kalman filters provide an accurate estimation of the first derivatives of their inputs as long as no noise is affecting the system.

The neural network to be trained was a feedforward neural network with 3 neurons in the hidden layer, no bias, and a total of 27 weights. This network was chosen because Igel [8] achieved the best performance with it. In the CMA-ES algorithm, the population sizes were chosen according to $\lambda = 4 + \lfloor 3\ln(n) \rfloor$, where n is the number of parameters to optimize, and the parent number was chosen to be $\mu = \lfloor \lambda/4 \rfloor$. The initial global-step size for CMA-ES was set to $\sigma_{(0)} = 1$ and the minimum to $\sigma_{min} = 0.05$.

Table 2 shows the results of learning to control the double pole balancing without velocities benchmark for different architectures. The result of including a Kalman filter as an estimator of velocity for each of the inputs increases the performance dramatically with an average of 302 evaluations ($std = 141$). This is almost 10 times faster than the best published performance to date. Moreover, the number of generalizations is also increased and outperforms the best results published so far, with an average of 334 ($std = 105$).

5.3 Robustness Against Noise

The second performance evaluation experiment makes complete use of the properties of the Kalman filter. Not only are the Kalman filter's estimation capabilities used, but also its proven ability to maintain a reliable estimate even under very noisy conditions. A Kalman filter is designed to take advantage of knowledge it has about the statistical nature of the noise present in the system. For this reason, the filter performs even better under noisy conditions than it does under zero-noise conditions.

The second experiment is similar to the first experiment, where the Kalman filter parameters are tuned so that the controller can pass the benchmark tests. The main difference in the second experiment is that a certain level of measurement noise is added to the system prior to tuning the parameters. The measurement noise that is introduced to the system is a Gaussian signal with zero mean and standard deviation $\sigma_v = 10^{-2}$. This noise is added to the inputs z_i of the Kalman filters depicted in Figure 2. The selected value of σ_v represents roughly 1.6% of the maximum amplitude of the pole positions and 0.41% of the maximum cart position. Although it may seem like a very small noise perturbance, one should take into account the following factors:

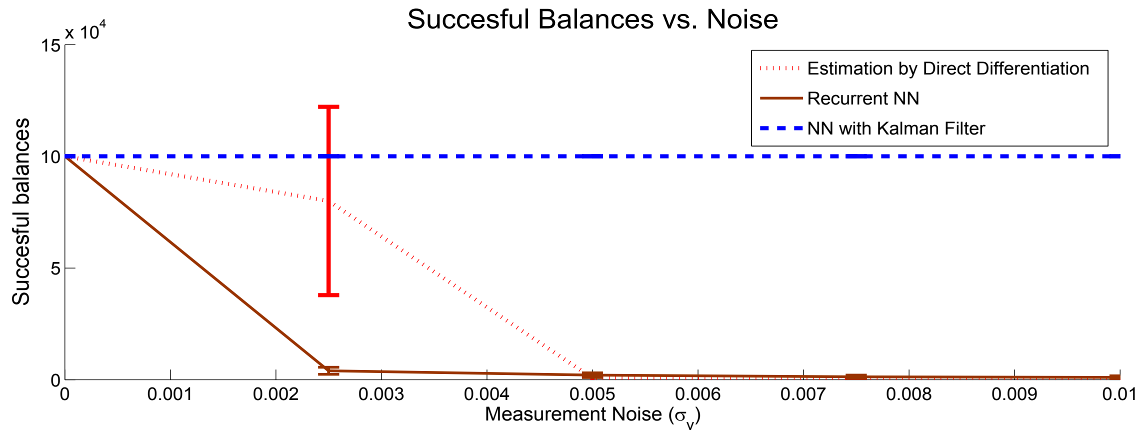
1. The nature of the problem: the double pole balancing problem is a controllable system, but its controllability is very small, and small perturbations or measurement errors make the system unstable. That is why the system is very likely to be successfully controlled in a simulation environment while not easily controlled in a real scenario.

2. The typical measurement noise magnitude: for this kind of scenario, it is assumed that an encoder reads angular positions with a nominal resolution of 1024 pulses per revolution. This finite resolution would lead to a measurement noise of 1/1024, ten times smaller than the measurement noise that we are introducing in our system. To check that this noise is sufficient to cause problems to other methods, we have compared our results to (a) a solution where a recurrent network is used, and (b) the same feedforward neural network that we have used with the Kalman filters, but replacing the filters with a direct numerical differentiation based velocity estimator.

In this experiment, the same neural network structure was used as in the first experiment, i.e. a feedforward neural network with 3 neurons in the hidden layer, no bias, and a total of 27 weights. Additionally, the optimization of the Kalman filters was incorporated into the evolutionary process, where optimal values for the parameters σ_{vi} and σ_{wi} were searched for using CMA-ES (along with the weights for the neural network). Because the problem is simulated, the standard deviation of the measurement noise we are introducing is known and thus the initial value for σ_{vi} in the Kalman filter can be set to this value. In the case of a real system, however, a set of real measurements could have been collected, the mean and standard deviation of the data set calculated, and the standard deviation used as the value for σ_{vi}. In the case of process noise, manual tuning is typically used due to the complexity of determining the value of the noise. The Kalman filter, however, usually performs well with only a rough estimate of σ_w.

The initial global-step size for CMA-ES is set to $\sigma_{(0)} = 1$ and the minimum is set to $\sigma_{min} = 0$. Since our evolutionary process is subject to random input noise, each individual is tested five times and an average fitness value is returned to the CMA-ES method, which then proceeds with the evolutionary process. The average number of evaluations obtained over 50 simulations was 1,354 ($std = 738$) and the average number of generalizations was 317 ($std = 91$).

Figure 4 depicts these results for increasing levels of measurement noise and for the following architectures: (1) the recurrent neural network solution (trained without noise), (2) the feedforward neural network with velocity estimation by direct numerical differentiation, and (3) the feedforward network with the Kalman filter. Each architecture is tested 10 times at each noise level. The results show the average and the standard deviation for each noise σ_v where $\sigma_v \in [0, 0.0025, 0.005, 0.0075, 0.01]$.



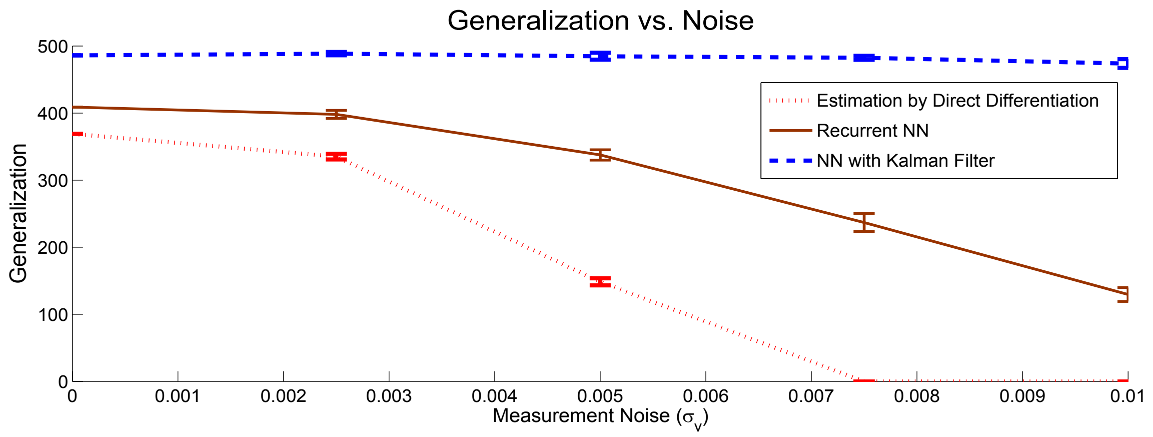


Figure 4: Results for different control architectures for the double pole balancing without velocities benchmark introducing measurement noise σ_v. Average over 10 simulations per noise value.

The upper figure shows the average number of successful balances out of a maximum of 10^5, as described in the first test of the benchmark problem. As can be seen, the feed-forward neural network with Kalman filtering achieves the maximum value of balances for all noise values. On the contrary, the recurrent neural network trained without noise has a very high sensitivity to noise and, for a value $\sigma_v = 0.0025$, the number of evaluations already drops to a value around 4000. Similarly, the feedforward neural network using a velocity estimation by numerical differentiation shows a better performance for low noise, but exhibits a high variance on the number of balances even when $\sigma_v = 0.0025$. For $\sigma_v = 0.005$, the number of balances drops to a value of 750. The bottom figure plots the average number of generalizations for each noise level. The feedforward neural network with Kalman filtering shows a relatively constant response to the different noise levels. Notice that the network and Kalman filters were trained for a noise level of $\sigma_v = 0.01$, so it is at this noise level that the network should exhibit the best performance. The network with velocity estimation by differentiation drops its performance by more than half by the time the noise level reaches $\sigma_v = 0.005$, and when $\sigma_v = 0.0075$ the number of generalizations has already gone to zero. On the other hand, although the recurrent neural network is not very successful in the first test, it has a quite remarkable performance in the second, still having an average of 130 generalizations with a noise level of $\sigma_v = 0.01$.

6. CONCLUSIONS AND OUTLOOK

Unlike the traditional recurrent neural network used for solving continuous state partially observable problems, we have proposed an alternative solution using a combination of Kalman filters and a feed-forward neural network. We have shown that it is possible to accelerate neuroevolutionary methods using the Kalman filter. Moreover, we have shown that the Kalman filter can be used to estimate the missing state variables. From the experimental results one can conclude that the evolved augmented neural network is robust against noise, which is a built-in feature of the Kalman filters used in the network. In the future, we plan to test our approach with other types of Kalman filters, and test the algorithm in other partially observable domains, particularly real world applications.

Acknowledgments

We would like to thank Nikolaus Hansen and Andreas Ostermeier for making the MATLAB implementation of the CMA-ES algorithm publicly available on the web. The implementation of the double pole balancing without velocities benchmark is based on the freely available source code by Faustino Gomez, Kenneth O. Stanley, and Risto Miikkulainen.

7. REFERENCES

[1] Y. Bar-Shalom, X. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, New York, USA, 2001.

[2] P. Dürr, C. Mattiussi, and D. Floreano. Neuroevolution with analog genetic encoding. In *Proceedings of the 9th Conference on Parallel Problem Solving from Nature (PPSN IX)*, pages 671–680, 2006.

[3] F. J. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[4] F. J. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning (ECML 2006)*, pages 654–662, 2006.

[5] F. Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Superieure de Lyon, Laboratoire de l'Informatique du Parallelisme, France, January 1994.

[6] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming: Proceedings of the First Annual Conference*, pages 81–89, Standford University, CA, USA, 1996. MIT Press.

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

[8] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Congress on Evolutionary Computation (CEC2003)*, volume 4, pages 2588–2595. IEEE Press, 2003.

[9] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] P. R. Kalata. Alpha-beta target tracking systems: A survey. In *American Control Conference*, pages 832–836, 1992.

[11] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, Series D:35–45, 1960.

[12] Y. Kassahun, M. Edgington, J. H. Metzen, G. Sommer, and F. Kirchner. A common genetic encoding for both direct and indirect encodings of networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 1029–1036, 7 2007.

[13] Y. Kassahun and G. Sommer. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 259–266, Bruges, Belgium, April 2005.

[14] L. J. Lin and T. M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science, USA, 1992.

[15] D. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

[16] N. Saravanan and D. B. Fogel. Evolving neural control systems. *IEEE Expert*, 3:23–27, 1995.

[17] H.-P. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[18] N. T. Siebel and G. Sommer. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems*, 4(3):171–183, 2007.

[19] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA, August 2004.

[20] G. Welch and G. Bishop. An introduction to the Kalman filter. Technical Report TR95-041, University of North Carolina at Chapel Hill, Department of Computer Science, USA, 1995.

[21] A. Wieland. Evolving controls for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, pages 667–673, 1991.