in: Australian Journal of Intelligent Information Processing Systems, 10 (2009) 3

Experimental Evaluation of Approaches for Long Term Prediction of Human Movement Trajectories

Sven Hellbach [1], Julian P. Eggert [2], Edgar Körner [2], and Horst-Michael Gross [1]
[1] Ilmenau University of Technology, Neuroinformatics and Cognitive Robotics Labs,
POB 10 05 65, 98684 Ilmenau, Germany
sven.hellbach@tu-ilmenau.de
[2] Honda Research Institute Europe GmbH, Carl-Legien-Strasse 30,
63073 Offenbach/Main, Germany
julian.eggert@honda-ri.de

Abstract. *This paper's intention is to adapt prediction algorithms well known in the field of time series analysis to problems being faced in the field of mobile robotics and Human-Robot-Interaction (HRI). The idea is to predict movement data by understanding it as time series. The prediction takes place with a black box model, which means that no further knowledge on motion dynamics is used then the past of the trajectory itself. This means, the suggested approaches are able to adapt to different situations. Several state-of-the-art algorithms such as Local Modeling, Cluster Weighted Modeling, Echo State Networks and Autoregressive Models are evaluated and compared. For experiments, real movement trajectories of a human are used. Since mobile robots highly depend on real-time application, computing time is also considered. Experiments show that Echo State Networks and Local Models show impressive results for long term motion prediction with a prediction horizon of up to eight seconds.*

1 Introduction

For autonomous social robots, like SCITOS [3], it is important to predict their own movement as well as the motion of people and other robots in their environment, for example to avoid collisions or to interact with moving people. Hence, further actions can be planned more efficiently. Most approaches in this field focus on optimal navigation strategies [6], [5]. This paper suggests spending more effort into prediction of the motion of the moving objects instead. Often, only linear approximations or linear combinations are used to solve this problem.

First of all, to be able to perform an adequate navigation, it is necessary to know the past motion trajectories of the surrounding dynamic objects. For simplification, a tracker is assumed, which is able to provide such trajectories in real-time. Scheidig [7] , for example, presents a person tracker, which provides the person's position and her motion trajectory projected onto the ground plane. Furthermore, it is possible to use a system tracking each of the person's limbs. This results in a complex trajectory in 3D space as it is shown in Figure 2. In both cases, the given trajectory of the motion can be interpreted as a time series $\mathcal{T}$ with values $\mathbf{s}_i$ for time steps $i=0,1,\ldots,n\text{-}1$: $\mathcal{T} = (\mathbf{s}_0,\mathbf{s}_1,\ldots,\mathbf{s}_{n-1})$.

For predicting the data coming from the tracker, an assortment of time series analysis algorithms has been implemented and comparatively tested.

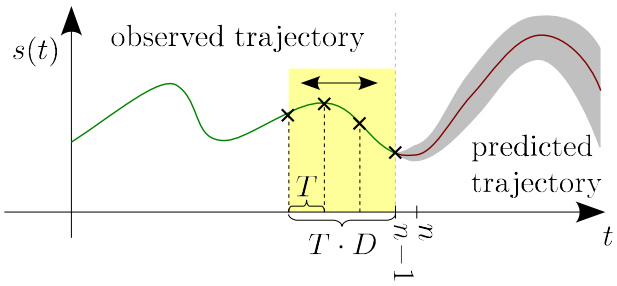


Figure 1: The observed trajectory (green) is to be predicted (red) for up to 500 time steps (about 8.3 sec. at 60 Hz). This is achieved only by exploiting the past trajectory's characteristics using a window (yellow) of D points equally spaced with interval T.

For applying these approaches with the described scenario, it is necessary that the algorithms fulfill some constraints for online application. First of all, calculation time is an important criterion. But, since most of the approaches rely on the trajectories' past, it has also to be considered, how much data is needed to generate a useful prediction. Observing a person for several seconds before being able to predict her motion, doesn't seem to be applicable. This problem is also discussed in the experiments, but the focus is to the prediction qualitiy.

The next section introduces our time series analysis approach to mobile robotics and HRI. Furthermore, the techniques are discussed, which are chosen to be tested. Section 2.1 discusses Echo State networks, which build their prediction by use of a randomly connected hidden layer, which is iteratively fed by the past trajectory points. Autoregressive Models from section 2.2 assume a linear relation in the time series which means that any time series value can be determined by using a linear combination of p previous values. Local Models try to find similar states of the observed trajectory (see Section 2.4). Similar to Local Models are Cluster Weighted Models (Section 2.5). However, for this approach the state space is clustered. In section 3 the comparing experiments with their conditions and results are presented, while the paper is concluded in Section 4.

2 Time Series Prediction

Our approach aims at the interpretation of movement data as time series to perform a long-term prediction. Within the field of time series analysis, a variety of algorithms

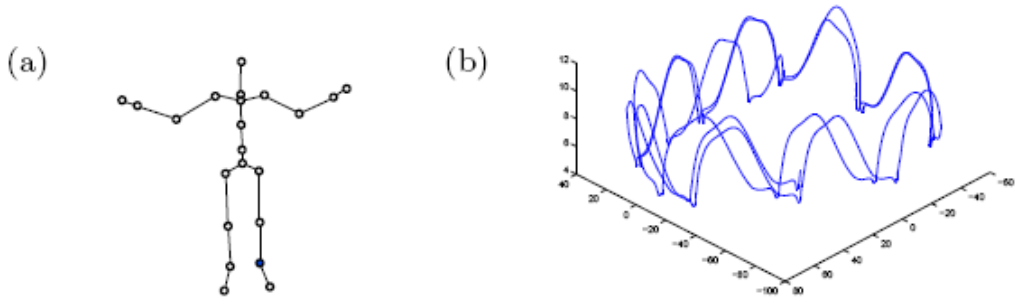


Figure 2: Example of movement data from the University of Glasgow. Shown are the body points from which data is available (a) and an exemplary trajectory of the movement of the left ankle while walking in circles (b).

does exist, coming from multiple applications with different backgrounds. Their fields of application reach from prediction of economic data to climate and biologic data, such as neural activities [1]. To provide a significant survey, state-of-the-art neural networks, probabilistic and deterministic approaches are selected for evaluation

The algorithms presented in this paper are intended to be used for motion prediction to enable a mobile robot a more anticipative navigation in dynamic environment. In such an environment the robots has to avoid collision with moving objects (humans, balls) or has to interact with them. For this purpose, the robot needs to predict the future movement of these dynamic objects. Basically, for all presented algorithms the prediction for each future point on the trajectory is done iteratively for up to 500 time steps (this corresponds to about 8.3 seconds of motion if using a sampling frequency of 60 Hz) (Figure 1). This is far beyond the abilities of a usual motion tracker, which only has to gap missing frames or deal with noisy sensor data.

The prediction in general takes place with the so-called black box model which means that no further background information is used than the past trajectory itself. The aspired prediction shall follow the trajectory's characteristics, which can be found in their past. Furthermore, no explicit trajectory models are given, to be able to freely adapt to completely new situations.

The output of the tracker is given as described in section 1. Hence, the $\mathbf{s}_i$ can be assumed as the tracked object's position, e.g. in a three dimensional Cartesian state space $\mathbf{s}_i=(x_i,y_i,z_i)^T$ (see Figure 2)

2.1 Echo State Networks

It is commonly known, that Neural Networks are well suited for function approximation tasks. For the specific task of predicting time series, Echo State Networks (ESNs) are often used in the recent years [4], [9].

2.1.1 Basic Idea

ESNs have some specific features which differ from "standard" neural networks: The hidden layer consists of neurons which are randomly connected (see Figure 3). If the connectivity is low, this layer provides independent output trajectories. For this reason, the hidden layer is also called reservoir. Furthermore, there are neurons which are connected to circles in the reservoir, so that past states "echo" in the reservoir. That is the reason why only the current time series value $\mathbf{s}_n$ is needed as input.

The weights of the reservoir determine the matrix $\mathbf{W}^r$. In [4], it is mentioned that the spectral radius $spec$ of this matrix[1] is an important parameter and must not have values above 1.0 to guarantee stable networks. The randomly initialized reservoir matrix $\mathbf{W}^r$ can easily be adapted to a matrix with a desired spectral radius. However, [9] argued that networks with a spectral radius close or slightly above 1.0 may lead to better results. Both possibilities are evaluated for their suitability for motion prediction.

Furthermore, the sparseness of the reservoir matrix plays an important role. A sparse reservoir matrix means that most of the weights in the reservoir are set to zero. This can be interpreted as the reservoir being decomposed into subsets, which are responsible for basic signals being overlaid by the output layer. As suggested in [4] and [9], about 80% of the weights are set to zero.

Another characteristic of ESNs is that only the output weights $\mathbf{w}_{out}$ are adapted and learned. All other weights (input, reservoir, feedback) are chosen randomly and stay static.

2.1.2 Training and Application

For training, the network is initialized randomly, and the training time series is used as network input step by step. The internal states $\mathbf{r}_n$ are calculated by using the following recursive equation:

$$\mathbf{r}_n = f(\mathbf{W}^r \cdot \mathbf{r}_{n-1} + \mathbf{w}_{in} \cdot \mathbf{s}_n + \mathbf{w}_{back} \cdot \mathbf{o}_{n-1})$$

$\mathbf{r}_n$ describes the internal state at time step n. $\mathbf{W}^r$ stand for the reservoir matrix, while $\mathbf{w}_{in}$ and $\mathbf{w}_{back}$ are the weights at the respective edges (see Figure 3), while f is the transfer function of the reservoir neurons, which can be the Fermi-function or the hyperbolic tangent.

From a predefined starting point, the internal states $\mathbf{r}_n$ can be combined to a matrix $\mathbf{R}$. The adaptation step for the output weights $\mathbf{w}_{out}$ is a linear regression using this matrix and the vector of the related output values $\mathbf{o}$:

$$\mathbf{w}_{out} = (\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\mathbf{o}$$

After weight adaptation, the network can be applied for prediction. Thereto, the network is fed again with the whole trajectory data as input, this time step by step. If the

[1] The spectral radius of a matrix equals to the largest Eigenvalue.

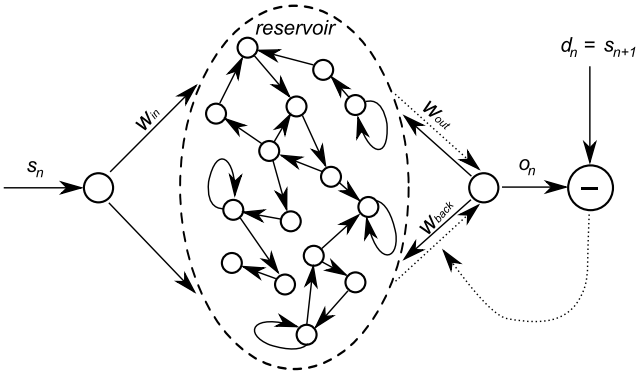


Figure 3: The design of Echo State Networks has some characteristic features. In addition to the randomly connected reservoir r_n, the training algorithm is pretty simple for neural networks: Only the output weights w_{out} are adapted.

prediction is taking place (i.e. reaching the last known point in time) the output is fed back to the input. So, the last network output is used as the next input to be able to generate more than one prediction step. In our experiments, up to 800 prediction steps are generated.

2.1.3 Enhancements

In [9] a few additional Echo State Network features are introduced, like an online adapting rule and a plasticity rule to adapt the Fermi transfer function parameters in the reservoir (*intrinsic plasticity*). Furthermore, additional weights such as a direct input-output ($\mathbf{w}_{dir}$) weight and a loop at the output neuron ($\mathbf{w}_{rec}$) are suggested. Apart from the online rule, all other of those enhancements were evaluated and tested.

Intrinsic plasticity is performed online. It helps to adjust the reservoir transfer functions for better adapting to the current prediction task. It takes place before starting the learning of the output weights and shouldn't last longer than 200 time steps, otherwise predictions could get instable. Unfortunately, intrinsic plasticity has the effect that the eigenvalues and thus the spectral radius of the reservoir matrix increases.

Since in Echo State Networks a huge number of parameters can be adjusted, a more automated process would be reasonable, especially, for those network weights, which are not changed during the regular training process, i. e. the $\mathbf{w}_{in}$, $\mathbf{w}_{back}$, and $\mathbf{W}^r$. We suggest to use multiple instances of the network, as a kind of simple stochastical search in the parameter space. All instances are trained using the same input data after initializing the fixed weights differently (in a random manner). During the training process, the output of each network is compared with the corresponding values of the training trajectory. The network showing the best prediction results for the yet unknown training data is then selected for further application.

2.2 Autoregressive Models

The next type of time series analysis algorithms introduced here are Autoregressive Models (AR). These models assume a linear relation in the time series which means that any time series value can be determined by using a linear combination of p previous values. The parameter p is the only one that needs to be defined. The general equation for calculation is as follows:

$$\mathbf{s}_n = \sum_{i=n-p}^{n-1} v_i \cdot \mathbf{s}_i$$

The v_i are the AR coefficients of the previous values and have to be calculated to predict future values. There are several possibilities to determine these coefficients. Three of them are discussed in the following.

2.2.1 Wiener Filter

The Wiener Filter simply does a linear regression with all points in the training time series [10]. In the literature usually the last p points are used. A matrix $\mathbf{E}$ with all points of the training time series and a vector $\mathbf{o}$ with the output values for each embedding point are generated. The AR coefficients arise from the linear regression:

$$\mathbf{v} = (\mathbf{E}^T\mathbf{E})^{-1}\mathbf{E}^T\mathbf{o}^T$$

Unlike as it is suggested in [10] to use only the last p values, the embedding discussed in section 2.3 is presented as input for the Wiener Filter. Experiments show that using the embedding leads to better results.

Note that this algorithm has similarities to the Local Modeling described in section 2.4 with the used regular embedding. However, the nearest neighbor search, needed for the Local Modeling algorithm, does not take place here.

2.2.2 Durbin-Levinson

This algorithm is based on the autocorrelation function (ACF) $\rho(h)$ which has to be determined before [8]. The ACF describes to some extend the self similarity of the given time series, by specifying the correlation of all possible delays between points on the time series. Then the AR coefficients can be calculated recursively to a defined AR depth p. Values around $p=100$ often lead to the best results.

In recursion step n the n-th coefficient v_n^n is computed. The upper index stands for the recursion step, while the lower specifies the coefficient.

$$v_n^n = \frac{\rho(n) - \sum_{k=1}^{n-1} v_k^{n-1}\rho(n-k)}{1 - \sum_{k=1}^{n-1} v_k^{n-1}\rho(k)}$$

All existing coefficients $v_1^{n-1},\ldots,v_{n-1}^{n-1}$ from the last recursion step are adapted as follows:

$$v_k^n = v_k^{n-1} - v_n^n v_{n-k}^{n-1} \quad k = 1, 2, \ldots, n-1$$

2.2.3 Yule-Walker

The last method for determining the coefficients uses the Yule-Walker equations [8]. For this approach, the ACF is needed again, but this time in its unnormalized version, as auto-covariance function $\gamma(h)$. Afterwards, the following equation system can be solved to get the AR coefficients:

$$\gamma(0) = v_1\gamma(1) + \ldots + v_p\gamma(p) + \sigma_w^2$$

$$\gamma(h) = v_1\gamma(h-1) + \ldots + v_p\gamma(h-p),\ h = 1, 2, \ldots, p$$

Unlike the Durbin-Levinson algorithm, now all coefficients are determined at once.

2.3 Embedding Space

For applying the approaches introduced in sections 2.4 and 2.5, an embedding in a higher dimensional space is necessary. The resulting trajectory can not be directly used as input for most of the discussed algorithms. Therefore, it has to undergo several preprocessing steps.

On the one hand, time series literature states that the mean has to be removed from the time series, and, therefore, from the trajectories. On the other hand, a higher dimensional embedding for the time series is generated. The well known sliding window approach can also be regarded an embedding. An observation window with size $T \cdot D$ is put on the trajectory (Figure 1). From this window, each T-th time step is used to generate the embedding. This kind of embedding is called *regular embedding* and two parameters are needed: the embedding delay T and the embedding dimension D. Such an embedding can be generated for each point of the trajectory from time step $T \cdot D$ on. The following equation shows the regular embedding $\mathbf{e}_t$ at time step t for time series values $\mathbf{s}_i$.

$$\mathbf{e}_t = \left(\mathbf{s}_t, \mathbf{s}_{t-T}, \mathbf{s}_{t-2T}, \ldots, \mathbf{s}_{t-(D-1)T}\right)^T$$

So, the time series is transformed into a D-dimensional space – the embedding space. To each embedding belongs an output $\mathbf{o}_t$, which stands for the successor $\mathbf{s}_{t+1}$ of the selected window.

The two introduced parameters T and D don't need to be defined by hand. Time series analysis offers techniques to automatically determine these parameters. For the calculation of the embedding delay T, the average mutual information function $I(T)$ is typically used. It is sufficient to search the first minimum in this function and to choose the right value for T as embedding delay [1].

$$I(T) = \sum_{\mathbf{s}_t, \mathbf{s}_{t-T}} P(\mathbf{s}_t, \mathbf{s}_{t-T}) \log_2 \frac{P(\mathbf{s}_t, \mathbf{s}_{t-T})}{P(\mathbf{s}_t) P(\mathbf{s}_{t-T})}$$

Discretized histograms from the observed time series are needed to calculate the probabilities $P(s_t, s_{t-T})$, $P(s_t)$, and $P(s_{t-T})$. Proper values for the number of histogram bins are between 15 and 30. In most cases, the smaller value is used to keep the calculation time low.

For the calculation of the embedding dimension D the embedding delay T is needed. Afterwards, so called true and false neighbors in embedding space need to be found. The idea behind the approaches for prediction is to find parts of the trajectory in their past, which are similar to the one from which prediction needs to approximated. Hence, a true neighbor is a real state space neighbor and qualified for a prediction. False neighbors only seem to be neighbors because of a too low embedding dimension. If the dimension is increased, what is equivalent with a further look into the past, different behavior will be observed from such false neighbors. To speed up the whole embedding procedure, not every embedding point is used for the classification in true and false neighbors, but a random selection of around 5% to 10% of the time series embedding points.

To find neighbors, in general a distance measure is necessary. In this paper, the Euclidian distance is used to separate true and false neighbors. The embedding dimension D is increased step by step starting with $D=1$. If a significant difference in a point's distance between D and $D + 1$ is observed, then this point is detected as a false neighbor. Otherwise this point seems to be a true neighbor. If the proportion of false neighbors falls below a certain value, then the associated value of D is used as embedding dimension.

To avoid this extensive search for parameters T and D, we, furthermore, used genetic algorithms instead. During the training phase, the best combinations of embedding dimension and embedding delay at a time are mutated. The prediction results from the subsequent algorithm (Local Modeling or Cluster Weighted Modeling) are used to determine which mutation works best.

2.4 Local Modeling

Local Modeling [2] is based on the aforementioned regular embedding. The principle idea is a simple nearest neighbor search in the embedding space of the last point in the time series $\mathbf{e}_{n-1}$ for which the prediction needs to be calculated. With the found nearest neighbors $\mathbf{e}_i$ and their corresponding outputs $\mathbf{o}_i$ the prediction is generated. "Near" means again a low distance in Euclidean space, but also other distance measures are possible.

In the general case, a polynomial is estimated for the prediction describing the relationship between embedding $\mathbf{e}_i$ and output $\mathbf{o}_i$. The nearest neighbors are used to define the polynomial's coefficients v applying linear regression. The influence of each neighbor can be controlled by a weight $\mathbf{w}_i$ depending on the distance to the embedding of the last point in time $\mathbf{e}_{n-1}$:

$$\mathbf{v} = (\mathbf{P}_W^T \mathbf{P}_W)^{-1} \mathbf{P}_W^T \cdot \mathbf{o}_W$$

Though, the weighted polynomial for each neighbor is represented in matrix $\mathbf{P}_W$ by putting the polynomial's monomials[2] into the matrix elements. $\mathbf{o}_W$ is the weighted vector of the neighbors outputs.

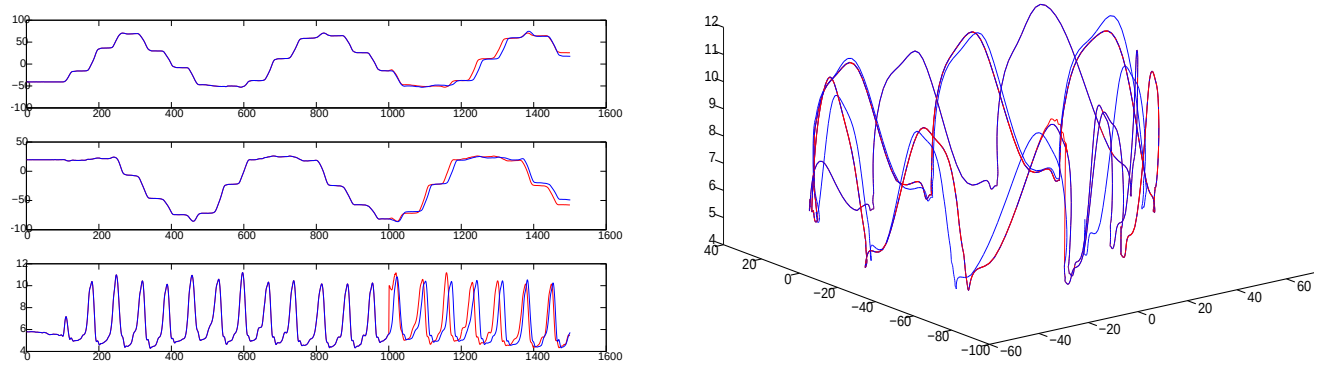
In practice, the polynomial degree g is usually low. Often its enough to use $g=0$ (*Local Averaging Model*) or $g=1$ (*Local Linear Model*).

In the first case, the polynomial simplifies to a constant and the linear regression to a weighted mean of the neighbors outputs:

$$\mathbf{v} = \mathbf{s}_n^{pred} = \frac{\sum_{i=1}^N w_i^2 \cdot \mathbf{o}_{n_i}}{\sum_{i=1}^N w_i^2}$$

Where $\mathbf{s}_n^{pred}$ is the prediction in time step n. The weights of the neighbors $\mathbf{w}_i$ can be determined in manifold ways. One example can be found in [2]. The basic idea is to rank the neighbors according to their distance. This means, that neighbors, which are far away, i.e. which belong to a less similar trajectory, have less influence.

[2] elementary part of the polynomial, consisting of only one term



(a) (b)

Figure 4: Prediction of the 3D movement trajectory from Figure 2 using Local Average Model. The prediction starts at time step 1000 and is calculated for 500 steps. No information about the actual trajectory is provided to the prediction algorithm from time step 1000 on. On the left side (a), the three dimensions are plotted over time. The right plot (b) shows a visualization in 3D space. The original trajectory is plotted in blue, while the prediction is in red color.

If the Local Linear Model is used, then the matrix $\mathbf{P}$ is filled with the embeddings of all N neighbors and with ones in the first column for the constant part of the polynomials. The polynomial is now a linear function depending on the embedding.

$$\mathbf{P} = \begin{pmatrix} 1 & \mathbf{e}_1 \\ \vdots & \\ 1 & \mathbf{e}_N \end{pmatrix}$$

After determining the coefficients v the prediction is calculated as follows:

$$\mathbf{s}_n^{pred} = v_0 + \sum_{i=1}^{D} \mathbf{e}_{n-1,i-1} v_i$$

Here v_0 stands for the constant of the polynomial.
To get good prediction results, it is crucial to choose proper parameters, such as the embedding parameters T and D and the number of the nearest neighbors N. Especially with higher polynomial degrees, the algorithm is extremely sensitive to the choice of these parameters. Therefore, an evolutionary algorithm was implemented which often leads to good results as recommended in [2].

2.5 Cluster Weighted Modeling

The Cluster Weighted Modeling, which is described also in [2], is operating in the embedding space, too. The viewpoint lies not on single embedding points like in the Local Modeling. Now the embedding space is clustered and covered with Gaussian representation. Each cluster c_m has a Gaussian representation $P(\mathbf{e}|\mathbf{c}_m)$ in the embedding space and another one in the output space $P(\mathbf{o}|\mathbf{e},c_m)$.
The so-called Input Domain $P(\mathbf{e}|c_m)$ specifies the membership from each cluster to the embedding for the last point in the time series which is to be predicted.

$$P(\mathbf{e}|c_m) = \prod_{i=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{m,i}^2}} \cdot e^{\frac{-(\mathbf{s}_i - \mu_{m,i})^2}{2\sigma_{m,i}^2}}$$

Note that the dimensions assumed to be independent to simplify the equation given above especially in high-dimensional spaces. Otherwise, a covariance matrix would be necessary [2].
The output terms $P(\mathbf{o}|\mathbf{e},c_m)$ determine the membership in the output space which is assumed to be one-dimensional, i.e. the next time step of the time series.

$$P(\mathbf{o}|\mathbf{e},c_m) = \frac{1}{\sqrt{2\pi\sigma_m^2}} \cdot e^{\frac{-(\mathbf{o}-f(e,\beta_m))^2}{2\sigma_m^2}}$$

The so called cluster function $f(\mathbf{e},\beta_m)$ represents the mean of the Gaussians. This function can be understood as similar to the polynomial in the Local Modeling (see section 2.4).
The following fraction can be understood as a weighted mean of the cluster function values $f(\mathbf{e}_n,\beta_m)$ of all clusters. The weighting takes place with the product from Input Domain $P(\mathbf{e}_n|c_m)$ and the general cluster weight $P(c_m)$.

$$\mathbf{s}_n^{pred} = \frac{\sum_{m=1}^{M} f(\mathbf{e}_n,\beta_m) P(\mathbf{e}_n|c_m) P(c_m)}{\sum_{m=1}^{M} P(\mathbf{e}_n|c_m) P(c_m)}$$

In these equations a lot of parameters need to be chosen: For each cluster, the mean and variance for the Gaussians in embedding space and output space is needed. Furthermore, the cluster function $f(\mathbf{e}_n,\beta_m)$ and the cluster weight $P(c_m)$ must be determined.
Typically an Expectation-Maximation-algorithm (EM-algorithm) is used to optimize most of the algorithm's parameters. This subroutine calculates in the E-step a a-posteriori distribution $P(c_m|\mathbf{o},\mathbf{e})$ of the existing data as follows:

$$P(c_m|\mathbf{o},\mathbf{e}) = \frac{P(\mathbf{o}|\mathbf{e},c_m) P(\mathbf{e}|c_m) P(c_m)}{\sum_{i=1}^{M} P(\mathbf{o}|\mathbf{e},c_i) P(\mathbf{e}|c_i) P(c_i)}$$

In the following M-step the parameters are updated. The algorithm can be found in detail in [2]. After several iterations, the EM-algorithm leads to a local minimum in the parameter space.
Only the number of clusters and the cluster function remain to be chosen manually. All other parameters are initialized randomly and adapted using the EM-algorithm. As cluster function, similar functions like the Local

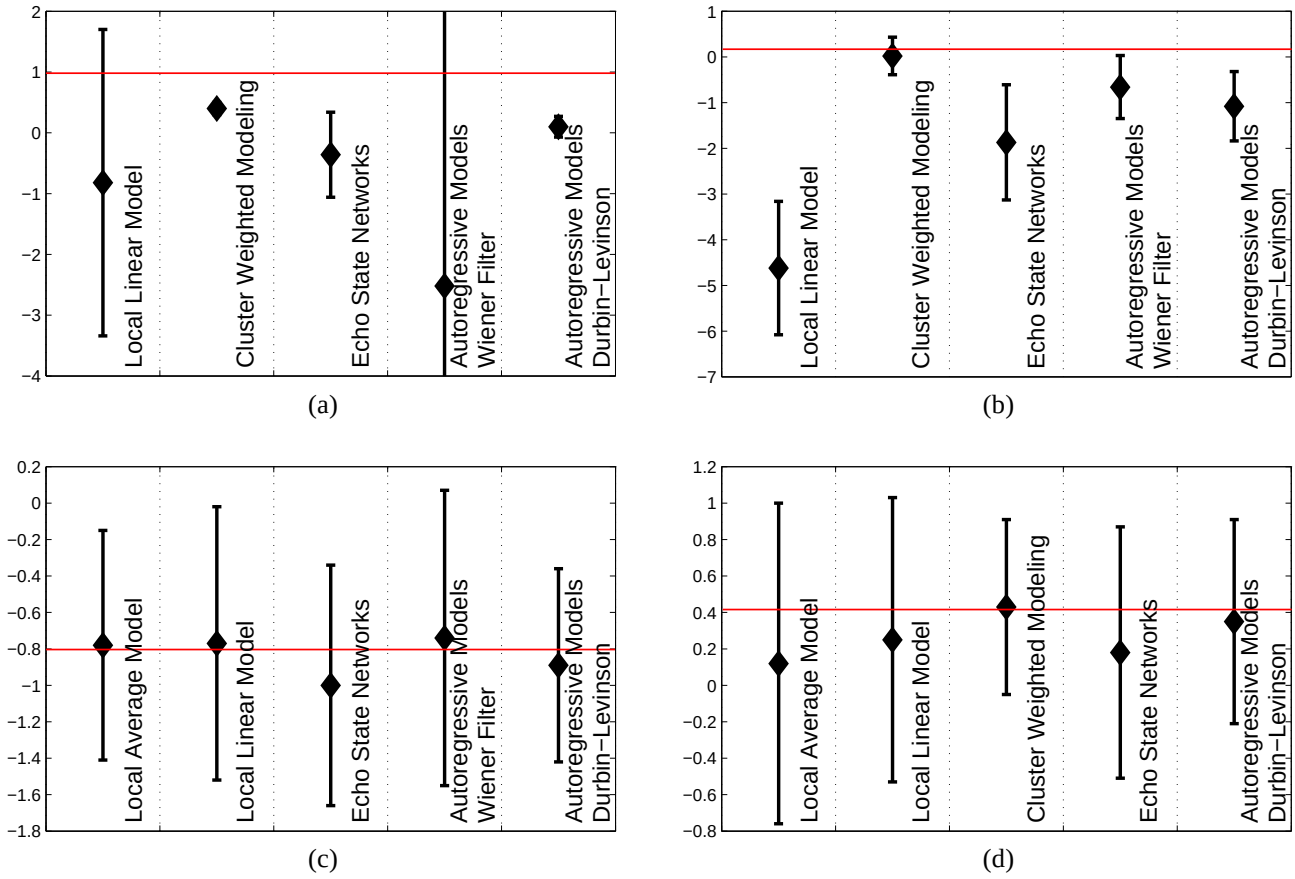


Figure 5: The graphs shows the STE (a), (c) and LTE (b), (d) plotted for each of the investigated algorithms tested with sine (a). Lorenz-attractor (b), and movement data (c), (d). The ordinate is scaled logarithmically. Hence, lower values mean a better prediction. The error bars represent the standard deviation from the mean. For the STE, all results lie relatively close together while the simple reference algorithm (red line), which is explained in section 3.1.2, can only be beaten clearly by the Echo State Networks. Longer predictions show more differences in the results of the algorithms. Also the mean errors are higher than STE, as being expected in longer predictions. The reference is beaten more clearly in general. Local Average Models (LAM) and Echo State Networks show the best results.

Modeling polynomials can be used. Since, calculation time strongly depends on the number of clusters, the value of these parameter should not be too high for an online application.

3 Motion Prediction

The algorithms presented in this paper are intended to be used for motion prediction to enable a mobile robot to navigate and to interact with humans in a dynamic environment. To be comparable and reproducible, movement data taken from the University of Glasgow[3] is used. This benchmark data is available as 3D coordinate representation for each limb of a human performing a certain action, e.g. walking (see Figure 2). Using this data is even more challenging, because several basic motions are combined (i.e. intrinsic movement, e.g. of the foot combined with the walking direction). The data set consists of 25 trajectories containing 1,500 up to 2,500 sampled points in Cartesian space. An example of a prediction of the 3D movement data using Local Linear Models (see Section 2.4), is shown in Figure 4.

[3] http://paco.psy.gla.ac.uk/data ptd.php

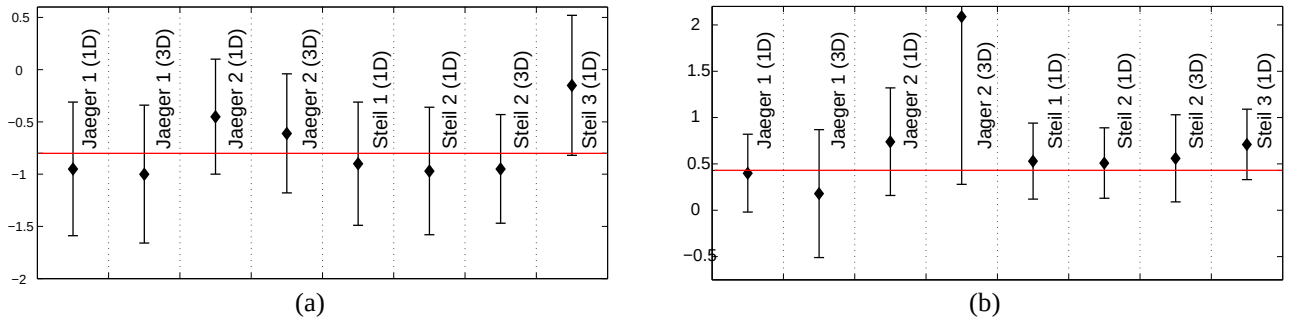
Besides the movement data coming from the University of Glasgow, periodical and "standard" chaotic time series are used. As chaotic time series the Lorenz-Attractor is used. It is a simple system of differential equations where the single dimensions are not independent. This time series is a typically chaotic one, so small changes in a state leads to huge differences after a short time period.
The periodical trajectories consist of up to three added sine waves, where each dimension is independent from the others.

3.1 Test Conditions

The movement data has a resolution of 60 time steps per second, so that an average prediction horizon of about 500 steps corresponds to a prediction of 8.3 seconds into the future. Most movement prediction techniques, as they are used for tracking, are designed to predict an objects position for the next time frame or at least to gap a loss of the object during a only a few frames.

3.1.1 Quality Measures

For comparing the prediction results, some kind of quality measures for comparison are necessary. The used quality measures are based on the normalized mean square error



(a) (b)

Figure 6: The graphs show the STE (a) and LTE (b) plotted for each of the Echo State Network version tested on 1D and 3D movement data. The ordinate uses a logarithmic scale. Hence, lower values mean a better prediction. The error bars represent the standard deviation from the mean. The different tests are labeled with "Jaeger" and "Steil" using the networks presented in [4] and [9] respectively. For Jaeger networks, two versions are tested. On the one hand, parameters, like number of neurons, spectral radius, and sparseness of the reservoir, where set to fixed values. On the other hand, those parameters are obtained randomly. For Steil networks, the number of neurons is increased (25, 100, 250). Additionally, version 3 of Steil network uses input s_{n-1} and s_{n-2} as input (not only s_{n-1} as for all other tests)

NMSE. Hence, the standard mean square error is normalized using the variance σ^2 of the time series.

$$NMSE = \frac{1}{N \cdot \sigma^2} \sum_{i=1}^{N} (\mathbf{s}_i^{pred} - \mathbf{s}_i^{orig})^2 = \frac{MSE}{\sigma^2}$$

Since the trajectories are three-dimensional and dimensions with greater difference are supposed to be more important, the highest variance of all dimensions is used as normalization.

Two different kinds of the defined measure are used. The first one, the short term error (STE), is responsible for evaluating a short period of the prediction. It uses the first $N = 75$ prediction steps (which means 1.25 sec) with a weighting of $1/f$ of the f-th prediction step. On the other hand, the performance is evaluated using the long term error LTE, which uses all prediction steps with a weighting of $1/\sqrt{f}$, since some of the algorithm show the tendency to drift away

3.1.2 Reference Algorithms

Additional simple reference algorithms were used which should be outperformed clearly to get a useful prediction. The first algorithm is a simple repetition of the last time series value. Also a linear approximation is used as reference, which simply does a linear approximation using the last two points of the time series. Both algorithms are tested on each data set. For a clear presentation of the experimental results in Figure 5 and Figure 6 the reference algorithm performing best is plotted as reference.

3.2 Results and Comparison

The following tests show the advantages and disadvantages of the different algorithms presented here. For the application, a number of parameters had to be decided to be able to apply the algorithms. The values used are chosen after extensive tests, which are not discussed here in detail.

The results for the experiments on the sine and Lorenz-Attractor data set are discussed at first. The purpose of these tests is the intention to stay comparable with time series analysis literature. Afterwards the results from the motion data set are presented.

3.2.1 Sine and Lorenz-Attractor trajectories

In the prediction of sine trajectories, the Wiener Filter shows the best results in the mean for STE and LTE (see Figure 5(a)). Note that Autoregressive Models can build up to high values, so that the standard deviation in this case is very high.

With worse mean errors the standard deviation is also lower. The Local Model and Echo State Networks lead also to quite good prediction results, while the reference is beaten clearly by all prediction algorithms, as expected.

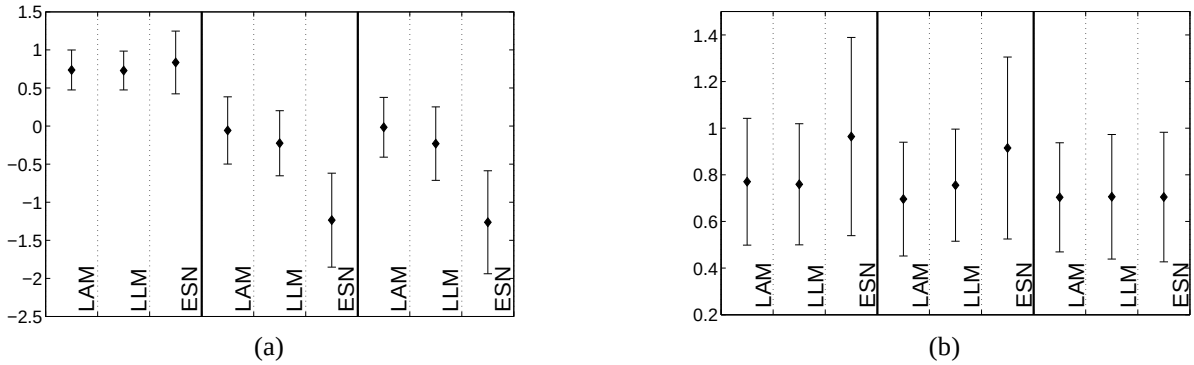
Predicting the chaotic Lorenz-Attractor (see Figure 5(b)) the Local Linear Models leads to the best results. Echo State Networks perform also well – especially with higher numbers of neurons. Here, the reference algorithms are outperformed clearly, as well. The standard deviation in the prediction quality is relatively high.

3.2.2 Real-world movement data

In the prediction of movement data, the Echo State Networks lead to the best results for the STE as it is shown in Figure 5(c), while for long term prediction Local Models have slightly better results (Figure 5(d)). The Autoregressive Models perform barely better than the reference. Here the Durbin-Levinson algorithm achieves the best prediction quality. Cluster Weighted Models show the worst performance and their mean errors stay even behind the simple reference algorithms. The best algorithms still beat the simple references clearly and are able to predict movements for several seconds (about 100 prediction steps) very well.

In general, the difference between each of the algorithms and the reference is much smaller than for the predictions of the sine or Lorenz-Attractor trajectories. Nevertheless, the best algorithms still beat the simple references clearly and are able to predict movements for several seconds (about 100 prediction steps) very well.

It can be assumed that the prediction of movement data is a harder problem than predicting standard chaotic trajectories, such as the Lorenz-Attractor. This is caused by unique unexpected and unpredictable behavior, which



(a) (b)

Figure 7: The graphs show the STE (a) and LTE (b) plotted for the most promising algorithms of the previous tests (Local Average Model (LAM), Local Linear Model (LLM), Echo State Networks ESN) in a similar fashion as in Figure 5. Each plot is separated into 3 sections. From left to right, these sections show the results for the test with the subsampled trajectory, the interpolated trajectory, and the comparison with the normal trajectory.

can be observed in the movement data. Therefore, the choice of the number of neurons in the Echo State Network reservoir for example has only a minor effect. In tests the difference in the prediction results of movement data between 25 and 250 neurons were insignificant. It can be presumed that the structure of the movement data does not allow a higher accuracy in the prediction unlike other chaotic time series [4].

3.2.3 A closer look on Echo State Networks

Since, Echo State Networks performed as one of the best algorithms for motion prediction, this section provides a closer look at different versions. The literature provides slightly different variants of Echo State Networks, two different ones are evaluated here. On the one hand, networks with a structure from [4], called in the following Jaeger networks and on the other hand, networks with a structure from [9] (Steil networks). For both networks, the spectral radius is set differently. While Jaeger [4] uses $spec=0.8$, with Steil networks it is set to $spec=1.0$.

Both networks are evaluated on real-world movement data (see Figure 6). As already mentioned, the movement data is available as a trajectory in 3D Cartesian space. These 3D points are used directly as input for the network (labeled "3D" in Figure 6), or they are split into three 1D time series, predicted independently with three networks (labeled "1D" in Figure 6).

It is recommended in [9] to initialize all Steil network weights to 0.05. Since only weights to the output layer are adapted during training process, all other weights stay at 0.05. Actually, this value could not be confirmed with the test on movement data. It could be shown for both network versions, that the feedback weights $\mathbf{w}_{back}$ must be scaled very low (about 10^{-20}) to guarantee stable networks. Furthermore, the input weights $\mathbf{w}_{in}$ are set to values of about 10^{-5}. For all other weights the influence of the chosen values is not that significant.

Steil networks have additional weights to the output layer ($\mathbf{w}_{dir}$,$\mathbf{w}_{rec}$). These weights can be included in the learning process as it is suggested in [9]. Unfortunately, in our experiments this leads to instable networks, so that these weights can not be learned for predicting the movement data. These weights should be scaled low about 10^{-20} as well, because they have a similar function like the feedback weights $\mathbf{w}_{back}$.

Jaeger [4] suggests to use 50 up to 2000 neurons for the reservoir. Because, a higher number of neurons doesn't lead to significant better results, the size of the reservoir is set to lie between 25 and 250 neurons.

Steil [9] advises to apply intrinsic plasticity (adaptation of transfer function parameters) for the first 200 time steps to improve classification results of the network. Those benchmark results in [9] were gained by applying the online learning rule. Since only offline learning rule is used here, the results could not be confirmed. In both types of networks, intrinsic plasticity seems to have only minor effects when the offline learning rule is applied.

Additionally, Steil networks were extended in a TDNN-like fashion, using more than only the last point of the trajectory as input (labeled "Steil 3" in Figure 6). It can be observed that this leads to better predictions in the very first steps (about 5 steps) but may destabilize the prediction in the following steps. Since the quality measures sum over 75 prediction steps, no improvement can be observed in the results.

3.2.4 Reduced set of training data

The evaluation discussed in the previous paragraphs used a time horizon of 1000 time steps for training. Towards online application, such a long training phase would require to observe the moving object for several seconds. Since, this is not possible in most cases, the tests depicted in Figure 7 are executed with less data. Only 300 time steps of the trajectory are used now. For the three left most results in Figure 7(a) and Figure 7(b), those 300 points in time are subsampled, as it would be the case when using a slow tracker. As it can be expected, the prediction quality significantly decreases (compared to the three right most results in Figure 7(a) and Figure 7(b)). A logical step at this point is to use interpolation to fill the missing gaps. For the test in Figure 7, a spline interpolation is used to gain 300 time steps of training data again. The results can be compared to the ones using the original trajectory (compare the three midway results in Figure 7(a) and Figure 7(b)).

Furthermore, it can be observed in general, that the results become worse than tested with a training period of 1000 points (Figure 5).

3.2.5 Calculation Time

For any online application, the calculation time is of high importance, since the movement is supposed to be predicted before it continues. Since, only MatLab implementations were tested on time series with lengths of around 1,000 till 2,500 time steps, only a first estimation can be given here.

Autoregressive Models and Echo State Networks with lower number of neurons show a calculation time of about 0.7 to 10 ms per prediction step. This is absolutely complying with online requirements.

Local Models and Cluster Weighted Models need longer calculation times between 50 and 250 ms. In the first case (Local Models), most calculation time is spend on the search for the nearest neighbors in the high number of training data. The Cluster Weighted Models are slow because of a long optimization time (the EM-algorithm).

As already said, the calculation times are only intended as a first estimation. For example, the nearest neighbor search for Local Modeling is implemented straight forward and, hence, quite time consuming. Nevertheless, it is clear to see, that Echo State Networks deserve further interest.

4 Conclusions and Future Works

The intention of this paper was to connect the well-known field of time series prediction and movement data handling from robotics or from human robot interaction in a consistent way. Different behaviors from the tested time series analysis algorithms were observed. Generally, it can be resumed that movement data behaves different than data from periodical and chaotic time series.

The tested algorithms show very good results in predicting several seconds of the movement data. Echo State Networks and Local Models pointed out to be suitable algorithms for movement prediction

Autoregressive Models and again Echo State Networks are able to predict fast enough for an online application without any further adaptation. From the current point of view, Echo State Networks are the “winning” approaches which are able to solve the problem best.

Local Models can be a good alternative to Echo State Networks if they could be accelerated without loss of quality. Besides this, enhanced versions of the Autoregressive Models such as ARMA or ARIMA Models could be tested. Furthermore, the usage of an irregular embedding is imaginable.

It could be shown that the quality of the prediction results strongly depends on the number of training data. On the other hand, reducing the number of training points is needed to go towards online application. Effort has to be spend to provide the approaches with the necessary amount of data. This data has to be provided in a smart way for not raising calculation time. For example, some cluster approaches allow to present the data to the Local Model in a way to speed up the nearest neighbor search.

In the introduction, it was mentioned to support a navigation task by the prediction. So, as a next step, it should be investigated, how the integration of the prediction into the navigation algorithm could be realized. One drawback for predicting movement data is the fact that human beings may perform unexpected motion. Since the discussed algorithms rely on the known characteristics, it is possible to use them for detection of such unexpected behavior.

5 Acknowledgments

Thanks to our students Sören Strauß and Sandra Helsper for doing the evaluation work and contributing helpful ideas.

6 References

[1] Abarbanel, H., Parlitz, U., “Nonlinear Analysis of Time Series Data”, In: Handbook of Time Series Analysis, WILEY-VCH, pp. 1 – 37, 2006

[2] Engster, D., Parlitz, U., “Local and Cluster Weighted Modeling for Time Serie Prediction”, In: Handbook of Time Series Analysis, WILEY-VCH, pp. 38 – 65, 2006

[3] Gross, H.M., Böhme, H.J., Schröter, C., Müller, S., König, A., Martin, C., Merten, M., Bley, A., “Shopbot: Progress in developing an interactive mobile shopping assistant for everyday use” In: Proc. IEEE Internat. Conf. on Systems, Man and Cybernetics (IEEE-SMC), pp. 3471-3478, 2008

[4] Jaeger, H., Haas, H., “Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication”, Science, pp. 78 – 80, April 2004

[5] Owen, E., Montano, L., “Motion planning in dynamic environments using the velocity space” In: Proc. of RJS/IEEE IROS, pp. 997 – 1002, 2005

[6] Pett, S., Fraichard, T., “Safe navigation of a car-like robot within a dynamic environment” In: Proc. of European Conf. on Mobile Robots, pp. 116 – 121, 2005

[7] Scheidig, A., Müller, S., Martin, C., Gross, H.M., “Generating person’s movement trajectories on a mobile robot”, In: Proc. of International Symposium on Robots and Human Interactive Communications (RO-MAN), pp. 747 – 752, 2006

[8] Shumway, R.H., Stoffer, D.S., “Time Series Analysis and Its Applications”, Springer Texts in Statistics, 2000

[9] Steil, J.J., “Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning”, Neural Networks 20, pp. 353 – 364, 2007

[10] Wiener:, N., “Extrapolation, Interpolation, and Smoothing of Stationary Time Series”, Wiley, 1949

7 Biography

Sven Hellbach is Ph.D. student at the Neuroinformatics and Cognitive Robotics Lab at Ilmenau Technical University since 2005. His research focus is set to motion analysis in the field of mobile robotics. The project, he is working for, is closely affiliated with

the Honda Research Institute Europe GmbH. He studied Computer Science at Ilmenau Technical University from 2000 to 2005.

Julian Eggert studied physics at the Technical University of Munich, Germany, where he also received his Ph.D degree in theoretical biophysics (Prof. van Hemmen) in 2000. In 1999, he joined the Honda Research Institute in Offenbach, Germany, concentrating on biophysically realistic large-scale models for vision systems. Since 2003, he worked as a Senior Research Scientist and since 2007 as a Chief Scientist heading a research division at the Honda Research Institute (HRI) Europe GmbH at Offenbach, Germany. His interests include probabilistic modeling of cognitive systems, perception models for dynamic scene interpretation and gating in hierarchical neural networks via feedback and attention.

Edgar Körner received his Dr-Ing in 1977 in the field of biomedical engineering, and his Dr. of Science (habilitation) in 1984 in the field of biocybernetics, both from the Technical University Ilmenau, Germany, where he became full professor and head of the department of neurocomputing and cognitive systems in 1988. From 1992 – 97 he was a chief scientist at Honda R&D Co. Wako, Japan. In 1997 he moved to Honda R&D Europe (Germany) to establish the “Future Technology Research Division”, and since 2003 he serves as the president of Honda Research Institute Europe GmbH at Offenbach, Germany. Since October 2007, he additionally serves as a co-director of the Research Institute for Cognition and Robotics at the University of Bielefeld. His research interest covers brain-like intelligence, with a special focus on self-referential control architectures, self-organization of knowledge representation, and autonomous robots.

Horst-Michael Gross is full professor of Neuroinformatics and head of the Neuroinformatics and Cognitive Robotics Lab at the Ilmenau University of Technology. He received his doctoral degree in Computer Science in 1989 from the Ilmenau University of Technology. Among his current research interests are neural computing, cognitive robotics, and multi-modal human-robot interaction in real world enviroments.