Appl Intell (2007) 26:99–109
DOI 10.1007/s10489-006-0007-1

Movement prediction from real-world images using a liquid state machine

Harald Burgsteiner · Mark Kröll · Alexander Leopold · Gerald Steinbauer

Published online: 13 November 2006

Abstract The prediction of time series is an important task in finance, economy, object tracking, state estimation and robotics. Prediction is in general either based on a well-known mathematical description of the system behind the time series or learned from previously collected time series. In this work we introduce a novel approach to learn predictions of real world time series like object trajectories in robotics. In a sequence of experiments we evaluate whether a liquid state machine in combination with a supervised learning algorithm can be used to predict ball trajectories with input data coming from a video camera mounted on a robot participating in the RoboCup. The pre-processed video data is fed into a recurrent spiking neural network. Connections to some output neurons are trained by linear regression to predict the position of a ball in various time steps ahead. The main advantages of this approach are that due to the nonlinear projection of the input data to a high-dimensional space simple learning algorithms can be used, that the liquid state machine provides temporal memory capabilities and that this kind of computation appears biologically more plausible than conventional methods for prediction. Our results support the idea that learning with a liquid state machine is a generic powerful tool for prediction.

Keywords Liquid state machine · Time series prediction · Recurrent spiking neural networks · Robotics

H. Burgsteiner
InfoMed/Health Care Engineering, Graz University of Applied Sciences, Eggenberger Allee 11, A-8020 Graz, Austria
e-mail: harald.burgsteiner@fh-joanneum.at

M. Kröll
Division of Knowledge Discovery, Know-Center, Inffeldgasse 21a, A-8010 Graz, Austria
e-mail: mkroell@know-center.at

A. Leopold
Institute for Theoretical Computer Science, Graz University of Technology, Inffeldgasse 16b/I, A-8010 Graz, Austria
e-mail: alexander.leopold@student.tugraz.at

G. Steinbauer (✉)
Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b/II, A-8010 Graz, Austria
e-mail: steinbauer@ist.tugraz.at

1 Introduction

The prediction of time series is an important issue in many different domains, such as finance, economy, object tracking, state estimation and robotics. The aim of such predictions could be to estimate the stock exchange price for the next day or the position of an object in the next camera frame based on current and past observations. In [7] decision-theoretic planning in combination with the agent-language GOLOG was used to control robots in a dynamic environment. Decision-theoretic planning uses predictions of the outcome of selected actions to derive the optimal plan. For the task of fault diagnosis in robot systems in general, models of the nominal behavior of the system over time are required. The outcome of these models are compared to the current state of the system in order to detect and identify faults in the system. In [26] particle filter techniques were used for this purpose. In the domain of robot control such predictions are used to stabilize a controller, to do planning with uncertainty and to automatically derive diagnosis about the robot's hard- and software. Jordan and Wolpert [13] provides a survey of different approaches in motor control where prediction enhances the stability of a controller.

There are two popular approaches for this kind of predictions: (1) modeling the behavior of the system or (2) learning prediction based on collected data. The first one claims a basic understanding of the system behind. It is preferred if

the internal structure of the system is well known and its behavior could be described sufficiently precise by a set of equations. In general this method is applicable to electronic circuits, technical processes or mechanical systems. A well known example for this approach is the prediction step in state estimation with Kalman-Filters [20]. It uses the current state and a linear system model to predict the state for the next time step. This prediction is optimal for linear systems. For non-linear systems the Extended Kalman-Filter (EKF) uses a linearization of the system. Hence, the EKF is not an optimal predictor anymore. The second approach is to learn the prediction from previously collected data. The advantages are that knowledge of the internal structure is not necessarily needed, arbitrary non-linear prediction can be learned and additionally some past observations could be integrated in the prediction. In this work we focus on the second approach.

Artificial Neural Networks (ANN) are common methods used for this type of computations. The common view of a neural network is that of a set of neurons plus a set of weighted connections (synapses in the biological context) between the neurons. Each neuron comes with a transfer function computing an output from its set of inputs. In multi-layer networks these outputs can again be used as an input to the next layer of neurons, weighted by the relevant synaptic "strength". *Feed-forward networks* only have connections starting from external input nodes, possibly via one or more intermediate hidden node processing layers, to output nodes. *Recurrent networks* may have connections feeding back to earlier layers or may have lateral connections (i.e. to neighboring neurons on the same layer). See Fig. 1 for a comparison of the direction of computation between a feed-forward and a recurrent neural network. With this recurrency, activity can be retained by the network over time. This provides a sort of memory within the network, enabling it to compute functions that are more complex than just simple reactive input-output mappings. This is a very important feature for networks that will be used for computation of time series, because current output is not solely a function of the current sensory input, but a function of the current and previous sensory inputs and also of the current and previous internal network states. This

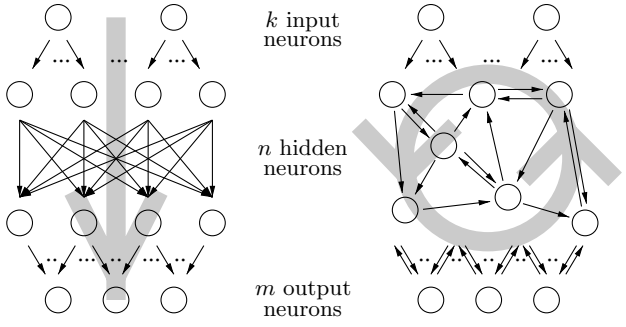

Fig. 1 Comparison of the architecture of a feed-forward (left hand side) with a recurrent neural network (right hand side); the gray arrows sketch the direction of computation

allows a system to incorporate a much richer range of dynamic behaviors. Many approaches have been elaborated on recurrent ANNs. Some of them are: dynamic recurrent neural networks [22], radial basis function networks (when one views lateral connections also as a recurrency) [2], Elman networks [5], self-organizing maps [14], Hopfield nets [11] and the "echo state" approach from [12].

In case of autonomous agents it is rather difficult to employ strictly supervised learning algorithms for recurrent ANNs such as backpropagation, Boltzmann machines or Learning Vector Quantization (LVQ), because the correct output is not always available or computable. It is also very difficult to set the weights of a recurrent ANN directly for a given non-trivial task. Hence, other learning techniques have to be developed for ANNs, that could simplify the learning process of complex tasks for autonomous robots.

Recently, networks with models of biologically more realistic neurons, e.g. spiking neurons, in combination with simple learning algorithms have been proposed as general powerful tools for the computation on time series [18]. In Maass et al. [17] this new computation paradigm, a so called *Liquid State Machine* (LSM)—which will be introduced in the next section—was used to predict the motion of objects in visual inputs. The visual input was presented to a 8×8 sensor array and the prediction of the activation of these sensors representing the position of objects for succeeding time steps was learned. This approach appears promising, as the computation of such prediction tasks is assumed to be similar in the human brain [1]. The weakness of the experiments in [17] is that they were only conducted on artificially generated data. The question is how the approach performs with real-world data. Real data, e.g. the detected motion of an object in a video stream from a camera mounted on a moving robot, are noisy and afflicted with outliers.

In this paper we present how this approach can be extended to a real world task. We applied the proposed approach to the RoboCup robotic-soccer domain. The task was movement prediction for a ball in the video stream of the robot's camera. Such a prediction is important for reliable tracking of the ball and for decision making during a game. The remainder of this paper is organized as follows. The next section provides an overview of the LSM. Section 3 describes the prediction approach for real data, Sections 4 and 5 cover the retrieval of this real data and the training of the prediction. Experimental results will be reported in Section 6 and discussed in Section 7. Finally, in Section 8 we draw some conclusions.

2 The liquid state machine

2.1 The framework of a liquid state machine

The LSM from [18] is the theoretical framework for computations in neural microcircuits. An exemplary structure is

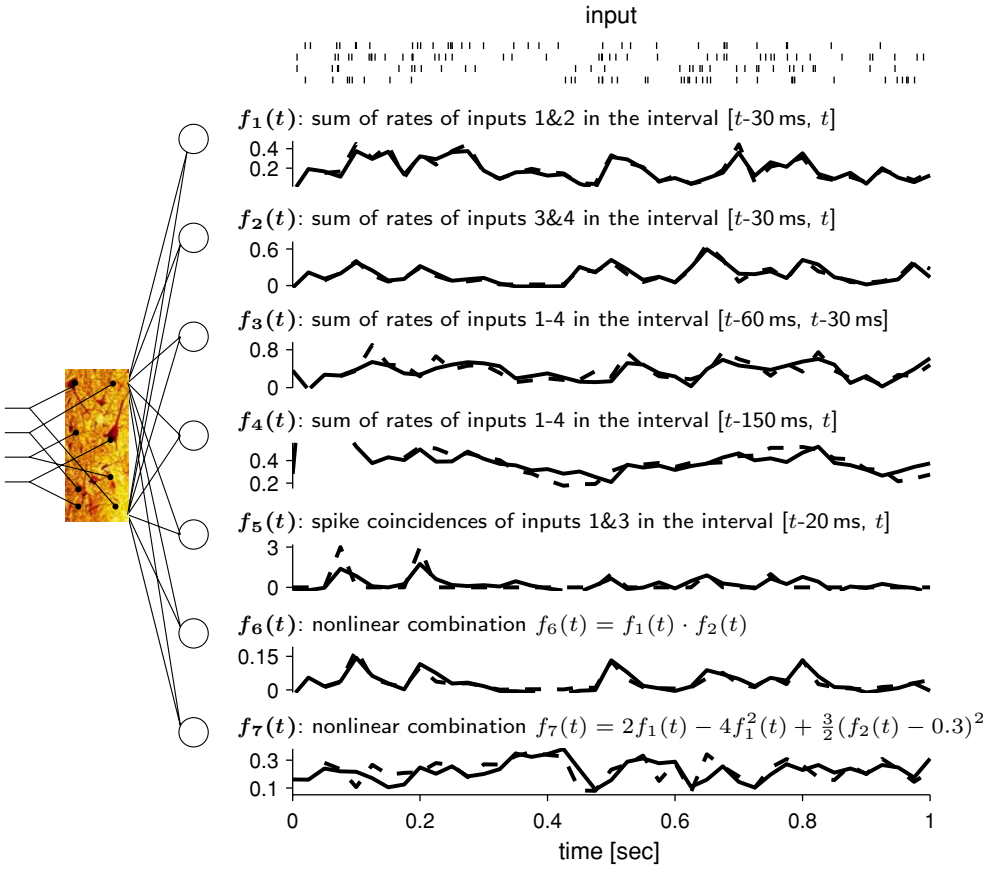

Fig. 2 Multi-tasking with any-time computing. A single neural microcircuit can be used by different readout-neurons to compute various function in parallel. In this case, based on a Poisson spike train as input to the LSM, 7 different functions were computed by readout neurons (figure from [21])

shown in Fig. 3. The term "liquid state" refers to the idea to view the result of a computation of a neural microcircuit not as a stable state like an attractor that is reached. Instead, a neural microcircuit is used as an *online computation tool* that receives a continuous input that drives the state of the neural microcircuit. The result of a computation is again a continuous output generated by readout neurons given the current state of the neural microcircuit.

Recurrent neural networks with spiking neurons represent a non-linear dynamical system with a high dimensional internal state, which is driven by the input. The internal state vector $x(t)$ is given as the contributions of all neurons within the neural microcircuit to the membrane potential of a readout neuron at the time t. The complete internal state is determined by the current input and all past inputs that the network has seen so far. Hence, a history of (recent) inputs is preserved in such a network and can be used for computation of the current output. For a detailed analysis of the properties of the LSM see [18]. The basic idea behind solving tasks with a LSM is that one does not try to set the weights of the connections within the neural microcircuit but instead only sets the weights of the readout neurons. This reduces learning dramatically and much simpler supervised learning algorithms which e.g. only have to minimize the mean square error in relation to a desired output can be applied. In fact, [6] have already used this principle to demonstrate that a simple bucket of water can also be used as the "liquid". They used the waves that are produced on the surface of a bucket full of water as the medium for a liquid state machine. The input was fed into the liquid with motors that perturbed the surface. A camera took pictures of the waves that originated that way. These digital images were the input to a simple perceptron that could solve the XOR-problem.

In [18] the LSM is also proven to have universal computational power on inputs of varying time series under idealized conditions. Furthermore, the LSM has several interesting features in comparison to other approaches with recurrent circuits of spiking neural networks:

1. The LSM provides "any-time" computing, i.e. one does not have to wait for a computation to finish before the result is available. Results start emitting from the readout neurons as soon as input is fed into the liquid. Furthermore, different computations can overlap in time. That is, new input can be fed into the liquid and perturb it while the readout still gives answers to past input streams.

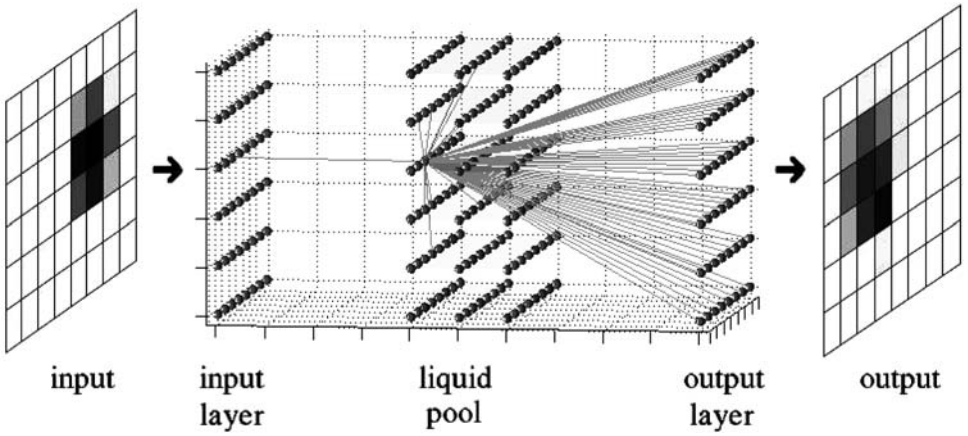

Fig. 3 Architecture of our experimental setup depicting the three different pools of neurons as well as sample input and output patterns with the data path overview. Example connections of a single liquid neuron are shown: input is received from the input sensor field on the left hand side and some random connection within the liquid. The output of every liquid neuron is projected onto every output neuron (located on the rightmost side). The $8 \times 6 \times 3$ neurons in the middle form the "liquid"

2. A single neural microcircuit can not only be used to compute a special output function via the readout neurons. Because the LSM only serves as a pool for dynamic recurrent computation, one can use many different readout neurons to extract information for several tasks in parallel. So a sort of "multi-tasking" can be incorporated. Figure 2 illustrates this and the previous property.
3. In most cases simple learning algorithms can be used to set the weights of the readout neurons. The idea is similar to kernel methods like support vector machines or kernel PCA, where one uses a kernel to project input data into a high-dimensional space. In this very high-dimensional space simpler classifiers can be used to separate the data than in the original input data space. The LSM has a similar effect as a kernel: due to the recurrency the input data is also projected into a high-dimensional space. Hence, in almost any case experienced so far simple learning rules like e.g. linear regression suffice.
4. The LSM is not only a computational powerful model, but it is also one of the biological most plausible so far. Thus, it provides a hypothesis for computation in biological neural systems.

2.2 Neural microcircuit

The model of a neural microcircuit as it is used for simulations in the LSM is based on biological evidence found in [9] and [25]. Still, it gives only a rough approximation to a real neural microcircuit since many parameters are still unknown. The neural microcircuit is the biggest computational element within the LSM, although multiple neural microcircuits could be placed within a single virtual model. In a model of a neural microcircuit $N = n_x \cdot n_y \cdot n_z$ neurons are placed on a regular grid in 3D space. The number of neurons along the x, y and z axis, n_x, n_y and n_z respectively, can be chosen freely. One also specifies a factor to determine how many of the N neurons should be inhibitory. Another important parameter in the definition of a neural microcircuit is the parameter λ. Number and range of the connections between the N neurons within the LSM are determined by this parameter λ. The probability of a connection between two neurons i and j is given by

$$p_{(i,j)} = C \cdot \exp^{-\frac{D_{(i,j)}}{\lambda^2}}$$

where $D_{(i,j)}$ is the Euclidean distance between those two neurons and C is a parameter depending on the type (excitatory or inhibitory) of each of the two connecting neurons. There exist 4 possible values for C for each connection within a neural microcircuit: C_{EE}, C_{EI},C_{IE} and C_{II} may be used depending on whether the neurons i and j are excitatory (E) or inhibitory (I). In our experiments we used spiking neurons according to the standard leaky-integrate-and-fire (LIF) neuron model that are connected via dynamic synapses. The time course for a postsynaptic current is approximated by the equation $v(t) = w \cdot e^{-\frac{t}{\tau_{syn}}}$ where w is a synaptic weight and τ_{syn} is the synaptic time constant. In case of dynamic synapses the "weight" w depends on the history of the spikes it has seen so far according to the model from [19]. For synapses transmitting analog values (such as the output neurons in our experimental setup) synapses are simply modeled as static synapses with a strength defined by a constant weight w. Additionally, synapses for analog values can have delay lines, modeling the time an action potential would need to propagate along an axon.

2.3 Training procedure

As it was said before, training a LSM requires only setting the weights of one or more output neurons. This output neuron

can in fact be any computational model that can map the state vector $x(t)$ to the desired binary or analog value. In our experiments perceptrons with a linear activation function in combination with linear regression as the supervised learning algorithm sufficed. A typical training procedure for a LSM includes the following steps:

1. Define a LSM with an appropriate size for the given task
2. Initialize the internal weights of the LSM to reasonable random values out of a given distribution
3. Feed the LSM with the training input $u(t)$ and record the internal state of the LSM $x(t)$ over time
4. Use any supervised training algorithm to compute the values of the weights of each readout neuron based on a given target vector $y(t)$.
5. Set the weights of the readout neuron(s) in the model to these values
6. Run the LSM with unseen test data and compare the performance

In experiments like in [3] this procedure can be extended by switching the LSM software into real-time mode and performing tasks with robots controlled by the trained LSM.

3 Experimental setup

In this section we introduce the general setup that was used during our experiments to solve prediction tasks with real-world data from a robot. As depicted in Fig. 3, such a network consists of three different neuron pools: (a) an input layer that is used to feed sensor data from the robot into the network, (b) a pool of neurons forming the liquid and (c) the output layer consisting of readout neurons which perform a linear combination of the membrane potentials obtained from the liquid neurons. These three parts together form the LSM.

For simulation within the training and evaluation the neural circuit simulator *CSim*[1] was used. Parameterization of the LSM is described below. Names for neuron and synapse types all originate from terms used in the *CSim* environment. Letters I and E denote values for inhibitory and excitatory neurons respectively.

To feed activation sequences into the liquid pool, we use *External Input Neurons* that conduct an injection current I_{inject} via *Static Analog Synapses* (parameters are shown in Table 1) into the first layer of the liquid pool. Inspired from information processing in living organisms, we set up a cognitive mapping from input layer to liquid pool. The value of I_{inject} depends on the value of the input data, in this case the activation of each single visual sensor.

[1] The software simulator *CSim* and the appropriate documentation for the LSM can be found on the web page `http://www.lsm.tugraz.at/`.

Table 1 Parameters for the static analog synapses which are used to feed input data into the LSM. 'EE' or 'EI' denotes whether the source and target neurons of a connection release excitatory or inhibitory action potentials, respectively. Covariance for $delay_{mean}$ is 0.1

	w_{mean}		$delay_{mean}$ $[ms]$	
I_{noise} $[nA]$	EE	EI	EE	EI
0	$3 \cdot 10^{-8}$	$6 \cdot 10^{-8}$	1.5	0.8

The liquid pool consists of *Leaky Integrate And Fire Neurons*—whose parameters are listed in Table 2—grouped in an $8 \cdot 6 \cdot 3$ cuboid, that are randomly connected via *Dynamic Spiking Synapses* (parameters are listed in Table 3), as described above. The probability of a connection between every two neurons is modeled by the probability distribution depending on a parameter λ described in the previous section. Various combinations of λ (average connection distance) and Ω (mean connection weight) were used for simulation. 20% of the liquid neurons were randomly chosen to produce inhibitory potentials. Figure 3 shows an example for connections within the LSM.

The information provided by the spiking neurons in the liquid pool is processed (read out) by *External Output Neurons* (V_{init}, $V_{resting}$, I_{noise} have the same values as for the liquid neurons), each of them connected to all neurons in the liquid pool via *Static Spiking Synapses* (parameters are listed in Table 4). The output neurons perform a simple linear combination of inputs that are provided by the liquid pool.

We evaluate the prediction approach by carrying out several experiments with real-world data in the Robo-Cup Middle-Size robotic soccer scenario. The experiments were conducted using a robot of the "Mostly Harmless" Robo-Cup Middle-Size team [8]. The task within the experiments is to predict the position of a ball in the field of view several frames into the future.

4 Generating input data

The input data was recorded by using a prototype of the middle-size-league RoboCup robot in use (and developed) by the RoboCup team at the Graz University of Technology. The experimental setup can be described as follows: the robot was located on the field and pointed its camera across the field. This robot tracked the movements of a soccer ball and featured a directional firewire camera driven by the *XVision* machine vision software [10], frequently delivering steady state images of 320×240 true color format. Time delays between transmission of two images varied from 70 ms to 200 ms. Similar to [16], the input to the LSM was provided by 48 sensors arranged in a 2D array (8×6), so the recorded images had to be preprocessed.

For each image, the x and y coordinates as well as the radius of the ball were extracted using an existing tracking

Table 2 Parameters for the leaky integrate and fire neurons comprising the liquid pool. Letters 'E' and 'I' indicate whether the neurons emit excitatory or inhibitory action potentials. $U(a, b)$ denotes an uniform distribution on the interval $[a, b]$

C_m $[nF]$	R_m $[M\Omega]$	V_{thresh} $[mV]$	$V_{resting}$ $[mV]$	V_{reset} $[mV]$	V_{init} $[mV]$	$T_{refract}$ $[ms]$ E	$T_{refract}$ $[ms]$ I	I_{noise} $[nA]$	I_{inject} $[nA]$
30	1	15	0	$U(13.8, 14.5)$	$U(13.5, 14.9)$	3	2	0	$U(13.5, 14.5)$

Table 3 Parameters for the dynamic spiking synapses connecting the neurons within the liquid pool. 'EE', 'EI', 'IE' and 'II' denote whether the source and target neurons of a connection emit excitatory or inhibitory action potentials. Covariance for $delay_{mean}$ is 0.1

Con.	U_{mean}	D_{mean}	F_{mean} $[s]$	$delay_{mean}$ $[ms]$	τ_{syn} $[ms]$	C
EE	0.5	1.1	0.05	1.5	3	0.3
EI	0.05	0.125	1.2	0.8	3	0.4
IE	0.25	0.7	0.02	0.8	6	0.2
II	0.32	0.144	0.06	0.8	6	0.1

Table 4 Parameters for the static spiking synapses connecting the read out neurons with each liquid neuron. 'EE' and 'EI' denote whether the source and target neurons of a connection fire excitatory (E) or inhibitory (I) action potentials. The covariance of $delay_{mean}$ is 0.1

τ_{syn} $[ms]$		$delay_{mean}$ $[ms]$	
EE	EI	EE	EI
3	6	1.5	0.8

package. The ball is detected within an image by simple color-blob-detection leading to a binary image of the ball. We can use this simple image preprocessing since all objects on the RoboCup-field are color-coded and the ball is the only red one. The segmented image is presented to the 8 times 6 sensor field of the LSM. The activation of each sensor is equivalent to the percentage of how much of the sensory area is covered by the ball. To group contiguous ball movements from the moment the ball entered the robot's field of view up to the point the ball left it (*movies*), we wrote an add-on for the XVision environment. Tracking information is stored line by line for each image containing coordinates, radius and time elapsed since start. Given this movie recording equipment, it was possible to record several hundreds of raw data movie files.

We collected a large set of 674 video sequences of the ball rolling with different velocities and directions across the field. The video sequences have different lengths and contain images in 50 ms time steps. These video sequences are transfered into the equivalent sequences of activation patterns of the input sensors. Figure 4 shows such a sequence. The activation sequences are randomly divided into a training set (85%) and a validation set (15%) used to train and evaluate the prediction. Training and evaluation is conducted for the prediction of 1 time step (50 ms), 2 time steps (100 ms) and 4 time steps (200 ms) ahead. The corresponding target activation sequences are simply obtained by shifting the input activation sequences 1, 2 or 4 steps forward in time.

5 Simulation and learning

Simulation for the training set is carried out sequence-by-sequence: for each collected activation sequence, the neural circuit is reset, input data are assigned to the input layer, recorders are set up to record the liquid's activity, simulation is started, and the corresponding recorded liquid activity is stored for the training part. The training is performed by calculating the weights of all static synapses connecting each liquid neuron with all output layer neurons using linear regression.[2] Let $\{m_{i,j}[n]\}$ be the activation sequence for sensor i out of the 8×6 sensor pool for one sequence j out of the training set. Let $\{x_i[n]\} = \{\{m_{i,1}\}, \{m_{i,2}\}, \ldots, \{m_{i,N}\}\}$ be the concatenation of all N activation sequences of the training set. With the expected output sequence $\{y_i[n]\} = \{x_i[n + p]\}$ consisting of the input sequence shifted by p prediction time steps, $\underline{w_i}$ as the weights of output neuron i and $\{\underline{psp[n]}\}$ as the sequence of postsynaptic potentials of all liquid neurons recorded during simulation, the regression writes as

$$\underline{w_i} = regress(\{y_i[n]\}, \{\underline{psp[n]}\})$$

for one neuron in the output layer. The least squares method is used for approximation. To get a more robust modeling, white noise was added to $\{\underline{psp[n]}\}$.

Analogous to the simulation with the training set, simulation is then carried out on the validation set of activation sequences. The resulting output neuron activation sequences are stored for evaluating the network's performance.

6 Results

We introduce the mean absolute error and the correlation coefficient to evaluate the performance of the network. The mean absolute error is the positive difference between the activation values of target and output sequences of the valida-

[2] In fact also the injection current I_{inject} for each output layer neuron is calculated. For simplification this bias is treated as the 0^{th} weight.

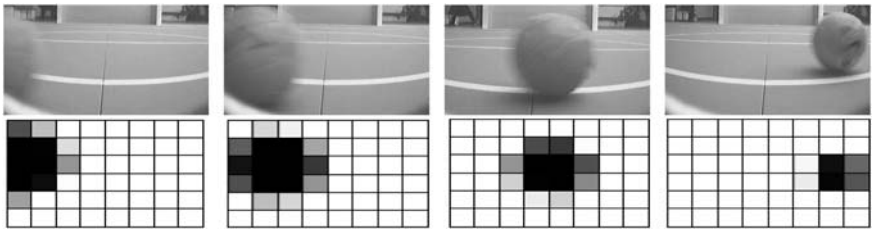

Fig. 4 Upper Row: Ball movement recorded by the camera. Lower Row: Activation of the input sensor field

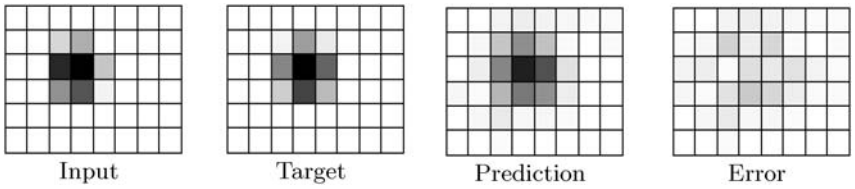

Fig. 5 Sensor activation for a prediction one timestep ahead. Input activation, target output activation, predicted activation and absolute error between target activation and predicted activation (left to right)

tion set divided by the number of neurons in the input/output layer and the length of the sequence. This average error per output neuron and per image yields a reasonable measure for the performance on validation sets with different length. Figure 5 shows an example for a prediction and its error.

A problem which arises if only this mean absolute error is used for evaluation is that also networks with nearly no output activation produce a low mean absolute error—because most of the neurons in the target activation pattern are not covered by the ball and therefore they are not activated leading to a low average error per image. The correlation coefficient measures the *linear* dependency of two random variables. If the value is zero two variables are not correlated. The correlation coefficient is calculated in a similar way as the mean absolute error. Therefore the higher the coefficient the higher the probability of getting a correlation as large as the observed value without coincidence involved. In our case a relation between mean absolute error and correlation coefficient exists. A high correlation coefficient indicates a low mean absolute error.

In Fig. 6 the mean absolute errors averaged over all single images in the activation sequences in the validation set and the correlation coefficients for the prediction one timestep (50 ms) ahead are shown for various parameter combinations. The parameter values range for both landscapes from 0.1 to 5.7 for Ω and from 0.5 to 5.7 for λ. If both Ω and λ are high, there is too much activation in the liquid. Remember, λ controls the probability of a connection and Ω controls the strength of a connection. This high activity hampers the network making a difference between the input and the noise. Both values indicate a good area if at least one of the parameters is low. Best results are achieved if both parameters are low (e.g. $\Omega = 0.5, \lambda = 1.0$). The figure clearly shows the close relation between the mean absolute error and the correlation coefficient. Furthermore, good results for the prediction can be observed for some parameter combinations. Maximum correlation coefficients are listed in Table 5.

We also compare the results achieved with two (100 ms) and four (200 ms) time steps predicted. In order to compare

Table 5 Maximum correlation coefficients by the LSM according to the desired prediction time

Prediction time	Correlation coefficient
50 ms	0.86
100 ms	0.74
200 ms	0.53

the results of both predictions for different parameter combinations, we use again a landscape plot of the correlation coefficients. Figure 7 shows the correlation coefficient for parameter values range from 0.1 to 5.7 for Ω and from 0.5 to 5.7 for λ. The regions of good results remain the same as in the one timestep prediction though lower correlation coefficients were achieved (about 0.7 at two timesteps and about 0.5 at four timesteps).

Not surprisingly the performance decreases when the task gets harder (i.e. the prediction time increases). Nevertheless, the results are good enough for reasonable predictions.

Figure 8 shows an example for the activations and the error for the prediction of two timesteps ahead. It clearly shows that the center of the output activation is in the region of high activation in the input and the prediction is reasonable good. Figure 9 shows that the activation is more and more blurred if the prediction time increases.

7 Discussion

In this section we discuss some observations that can help to give answers to questions such as "Why do certain parameter combinations (Ω, λ) yield better results than others?", "How come a recurrent neural network is capable of predicting something?" and "Will the precision of the prediction raise with the network size?". Moreover the LSM is exhibiting similar effects as kernel functions used in kernel methods, like kernel PCA. Thus, we discuss the pros and cons of both approaches. This section is completed by reasoning about real-time applicability of the LSM.

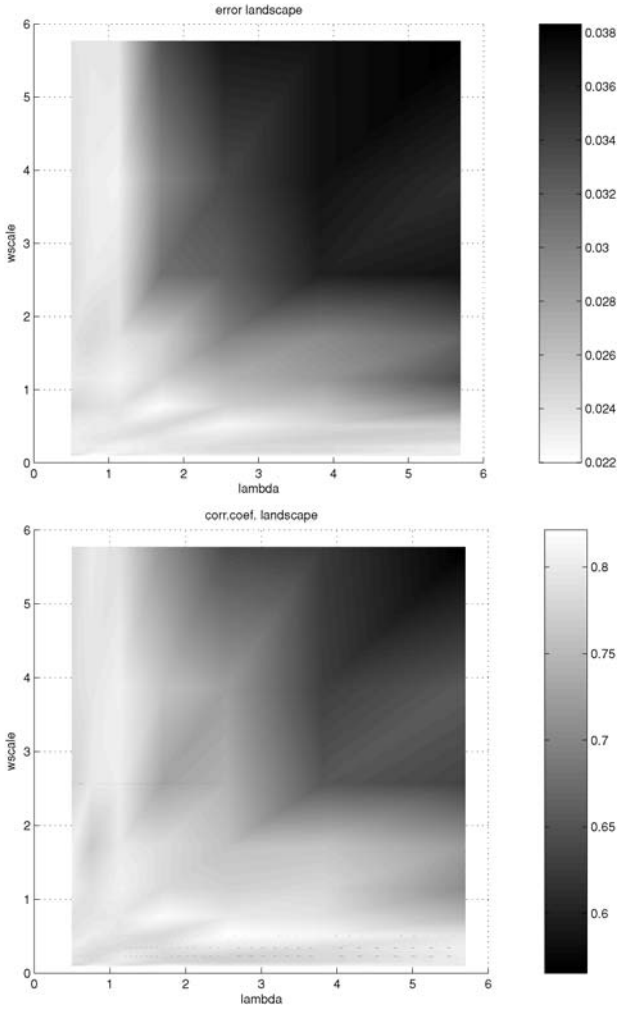

Fig. 6 Mean absolute error landscape (upper plot) and the corresponding correlation coefficient landscape (lower plot) for a prediction of one time step (50 ms) ahead. Mean connection weight $\Omega(wscale) \in [0.1, 5.7]$, average connection distance $\lambda \in [0.5, 5.7]$

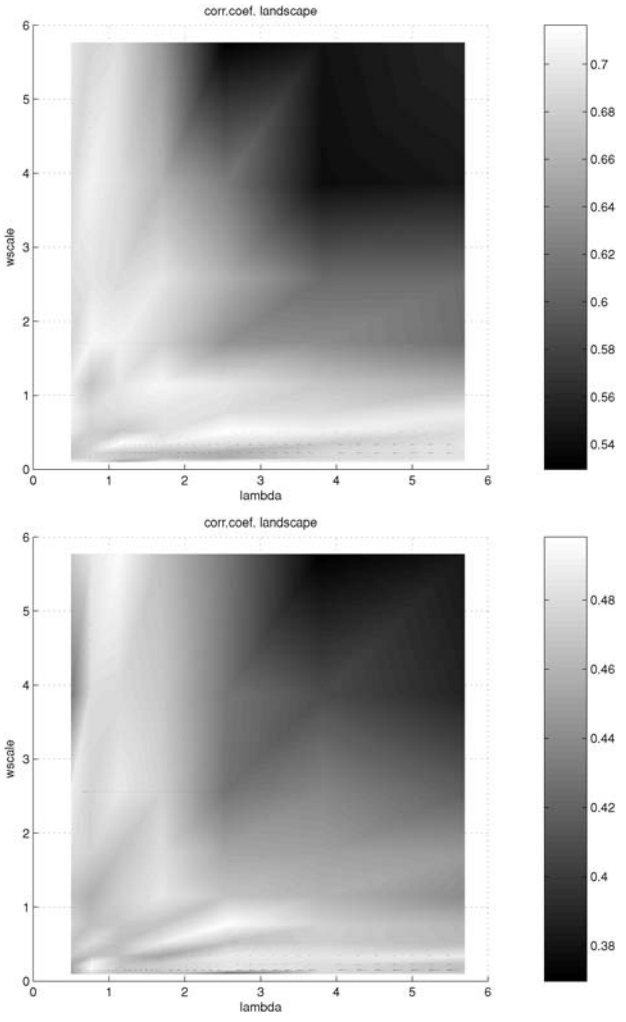

Fig. 7 Correlation coefficient landscape for a prediction of two timesteps (100 ms) ahead on the upper plot and of four timesteps (200 ms) on the lower plot. Mean connection weight $\Omega(wscale) \in [0.1, 5.7]$, average connection distance $\lambda \in [0.5, 5.7]$

7.1 Edge of chaos

Certain parameter combinations (e.g. the light shaded region in Fig. 6) yield better results than others. In [15] it is shown that cortical microcircuits do operate at the edge of chaos, a region that is located at the boundary between ordered and chaotic behavior. It turns out that in the study of neural systems this research direction is of special interest, since dynamic systems exhibit enhanced computational power in this region.

At this critical line, the antagonistic effects of the fading memory property [23] and the separation property [17] reaches an equilibrium state. The ordered phase is typically characterized by the fading memory property—small differences in the network state tend to decrease rapidly over time. In chaotic networks these differences are highly amplified and do not vanish. In the landscape plots we presented in Section 6, regions can be spotted whose parameter combinations yield optimal performance according to the task (see Table 5). This region of highest correlation coefficients does not change throughout the various prediction tasks. In our understanding the network operates at the edge of chaos when initialized with the corresponding parameter combinations. Similar parameter regions are obtained in Legenstein et al. [15] when performing a spike train classification task using a LSM.

7.2 Temporal integration

The task of predicting ball movements from simple 2-D images asks for some kind of memory. Hence, the neural microcircuit needs to feature the proper degree of *temporal integration*. According to [21] the temporal memory is influenced significantly by the number of neurons comprising the liquid and the type of synapses used for connecting these neurons. Thus by increasing the network size a better tem-

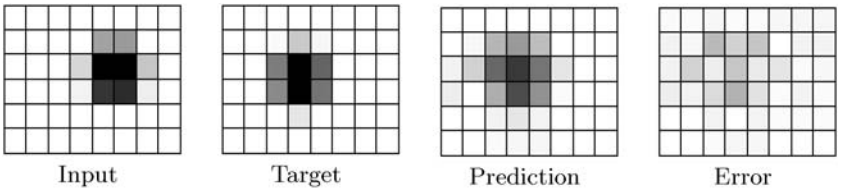

Fig. 8 Sensor activation for a prediction two timesteps ahead. Input activation, target output activation, predicted activation and absolute error between target activation and predicted activation (left to right). Parameters: Mean connection weight $\Omega = 1.0$, average connection distance $\lambda = 2.0$

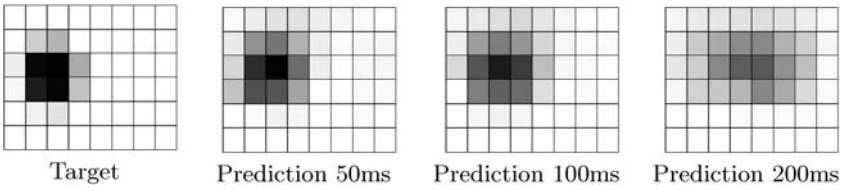

Fig. 9 Target output activation for some frame and corresponding predicted activations when predicting 1, 2 and 4 timesteps ahead (left to right)

poral integration ability is achieved. Nevertheless we keep at a network size of 144 neurons in the liquid not only because of the desired real-time applicability but also because of too little performance improvement discussed in 7.3. An important factor is the usage of dynamic synapses as it is reported in [21] yielding superior performance to static synapses.

7.3 Size and topology

As mentioned in Section 2, the network acts similar to kernel functions of support vector machines due to its recurrency. The input is projected into a high-dimensional state within the neural microcircuit. Therefore a network state consists of an abundance of non-linear combinations. The dimensionality of that particular state depends on the number of neurons in the network and the density of the connections between the neurons. Clearly, if the network is too small, the non-linearity that is hidden in the input sequence cannot be contained in the network state and therefore not revealed by linear learning rules. On the other hand if the network is too large, the main part of non-linear combinations in the network state is redundant and a significantly better performance is not achieved.

We carried out the same experiments we reported in the previous sections with network sizes of 36 and 576 liquid neurons. In the first case the results were unsatisfactory and in the latter case the resulting correlation coefficients exceeded only marginally the ones with 144 neurons. Thus a network size of 144 neurons in the liquid is sufficient for this prediction task.

7.4 Comparison with a kernel method

We also carried out this movement prediction task employing a suitable kernel method. We selected Kernel Principal Component Analysis (kernel PCA) [24], since it is a powerful technique for extracting non-linear structure from input data that is mapped into a high dimensional space in order to perform PCA in that space.

The same raster input that is fed into the LSM is rearranged into vector form to be able to apply a kernel function, the basic component in all kernel methods. It can be described as a "comparison function" $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ where $\mathcal{X}$ denotes the input data set. The value that is returned by $k(x, x')$ (where $x, x' \in \mathcal{X}$) is large when x and x' are "similar". We make use of a radial basis function kernel to obtain a Gram matrix (kernel matrix) that is then diagonalized to get an eigenvalue decomposition. Eigenvalues and corresponding eigenvectors are then sorted in descending order beginning with the largest. By projecting the transformed input data onto the eigenvectors, principal components are obtained in an unsupervised manner. A simple linear learning algorithm (linear regression) is carried out to calculate coefficients that map the principal components to the desired target data. In the evaluation stage, the same kernel function is applied to the validation data to obtain again a kernel matrix. The transformed validation data are projected onto the eigenvectors that were acquired during the training stage. By means of the extracted features and the former learnt coefficients a prediction of the ball position is calculated.

The correlation coefficient is calculated to evaluate the performance (see Table 6) and to be able to compare the results with the LSM approach. The training set comprises 180 activation sequences and the validation set 100 activation sequences. Both sets were chosen randomly out of the original 674 sequences. The number of used features depends on the decrease of the eigenvalues - eigenvectors whose eigenvalues are close to zero are not taken into account (in order to separate information from noise). 40 features sufficed for the linear regression. The RBF kernel width was set to 0.95 to achieve largest correlation coefficients.

The kernel method performs worse than the LSM approach, mainly because it lacks temporal integration. There

Table 6 Comparison of maximum correlation coefficients achieved by kernel PCA and the LSM

	Correlation coefficients	
Prediction time	kernel PCA	LSM
50 ms	0.86	0.86
100 ms	0.67	0.74
200 ms	0.43	0.53

is only information about the ball position in one input vector, information about the ball trajectory is missing. Moreover the number of training sequences is limited by the kernel method itself. Because of those computational constraints only 180 sequences were used for training resulting in matrices of approximately 4000×4000 elements that have to be processed. That limitation is disadvantageous compared to the LSM, since not all possible input variations occur during the training period. Furthermore kernel PCA can only be applied offline, whereas the LSM provides *any-time* computing, thus as soon as input is fed into the network the corresponding output is available.

7.5 Real-time applicability

In order to utilize the LSM not only in real world *data* applications but also in real *time* applications, one needs a simulation framework that provides the proper interface as well as sufficient simulation speed. In [3] the real-time extensions of the LSM simulator are used successfully for imitation learning experiments with a Khepera miniature robot. The work introduces a simple extension enabling real-time experiments. This of course only works for network sizes and activities that can be simulated in a time smaller than the time that is passing in the simulated neural network. This constraint is not too hard to meet for network sizes like we used in our experiments. E.g. our simulations of the networks with 144 neurons were performed on a desktop PC with a 1.9 GHz Intel® Pentium® 4 CPU and 512 MB RAM and took place in about $\frac{1}{4}$ of real-time. This framework enables us to implement a much richer range of experiments that can use noisy real world data on agents that operate in real-time. We are currently working on a goal keeper where we use the ball prediction from this article to control a robot that is able to intercept a ball. For a more detailed discussion of the real-time framework of the LSM see [4].

8 Conclusion

In this work we propose a biologically realistic approach for the computation of time series of real world images. The *Liquid State Machine* (LSM), a biologically inspired computation paradigm, is used to learn ball prediction within the RoboCup robotic soccer domain. The advantages of the LSM are that it projects the input data into a high-dimensional space and therefore simple learning methods, e.g. linear regression, can be used to train the readout. Furthermore, the *liquid*, a pool of inter-connected neurons, serves as a memory which holds the current and some past inputs up to a certain point in time (temporal memory). Finally, this kind of computation is also biologically more plausible than other approaches like Artificial Neural Networks or Kalman-Filters.

Experiments within the RoboCup domain show that the LSM approach is able to reliably predict ball movement up to 200 ms ahead. The experimental setup for this task and the corresponding results are presented. These results support the idea of the LSM as a generic powerful prediction mechanism for time series.

Furthermore, a deeper discussion was made about philosophy behind the liquid state machine, the necessary topologies, sizes and its real-time applicability. Moreover, a systematic comparison to the kernel PCA method was made to show that the more general LSM approach performs similarly as less general methods.

References

1. Bear MF (2000) Neuroscience: exploring the brain. Williams and Wilkins, Baltimore, MA
2. Bishop C (1995) Neural networks for pattern recognition. Oxford, UK, Oxford University Press
3. Burgsteiner H (2005) Training networks of biological realistic spiking neurons for real-time robot control. In: Proceeding of the 9th International Conference on Engineering Applications of Neural Networks, pp 129–136
4. Burgsteiner H (2006) Imitation learning for real-time robot control. Int J Eng Appl Artif Intell 19:741–752
5. Elman J (1990) Finding structure in time. Cognitive Sci 14:179–211
6. Fernando C, Sojakka S (2003) Pattern recognition in a bucket: a real liquid brain. In: Advances in Artificial Life: 7th European Conference, Lecture Notes in Computer Science, vol. 2801, Springer, Berlin/Heidelberg, pp 588–597
7. Ferrein A, Fritz C, Lakemeyer G (2004) On-line decision-theoretic Golog for unpredictable domains. In: Proc. 4th Cognitive Robotics Workshop at ECAI 04
8. Fraser G, Steinbauer G, Wotawa F (2004) A modular architecture for a multi-purpose mobile robot. In: Innovations in Applied Artificial Intelligence, IEA/AIE, Lecture Notes in Artificial Intelligence, vol. 3029, Springer, Canada
9. Gupta A, Wang Y, Markram H (2000) Organizing principles for a diversity of gabaergic interneurons and synapses in the neocortex. Science 287:273–278
10. Hager G, Toyama K (1998) The XVision system: a general purpose substrate for portable real-time vision applications. Comput Vis Image Underst 69:23–37
11. Hopfield J (1982) Neural networks and physical systems with emergent collective computational abilities. In: Proceeding of the National Academy of Science 79:2554–2558
12. Jaeger H (2001) The echo state approach to analysing and training recurrent neural networks. Tech Rep 148, GMD

13. Jordan M, Wolpert D (1999) Computational motor control. In: Gazzaniga M (ed) The Cognitive Neurosciences. Cambridge, MA, MIT Press
14. Kohonen T (2001) Self-Organizing maps, 3 edn. Springer-Verlag
15. Legenstein R, Maass W (2005) What makes a dynamical system computationally powerful? In: Haykin S, Principe JC, Sejnowski T, McWhirter J (eds) New Directions in Statistical Signal Processing: From Systems to Brain. MIT Press.
16. Legenstein RA, Markram H, Maass W (2003) Input prediction and autonomous movement analysis in recurrent circuits of spiking neurons. Reviews in the Neurosciences (Special Issue on Neuroinformatics of Neural and Artificial Computation) 14(1–2):5–19
17. Maass W, Legenstein RA, Markram H (2002) A new approach towards vision suggested by biologically realistic neural microcircuit models. In: Buelthoff HH, Lee SW, Poggio TA, Wallraven C (eds) Biologically Motivated Computer Vision. In: Proc. of the Second International Workshop, BMCV 2002, Lecture Notes in Computer Science, vol 2525, Springer, Berlin, pp 282–293
18. Maass W, Natschlaeger T, Markram T (2002) Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Comput 14(11):2531–2560
19. Markram H, Wang Y, Tsodyks M (1998) Differential signaling via the same axon of neocortical pyramidal neurons. In: Proceedings of the National Academy of Science 95(9):5323–5328
20. Maybeck PS (1990) The kalman filter: an introduction to concepts. In: Cox I, Wilfong G (eds) Autonomous robot vehicles, Springer-Verlag, pp 194–204
21. Natschläger T, Markram H, Maass W (2004) Computational models for generic cortical microcircuits. Computational Neuroscience: A Comprehensive Approach, pp 575–605
22. Pearlmutter B (1995) Gradient calculation for dynamic recurrent neural networks: a survey. IEEE Transactions on Neural Networks 6(5):1212–1228
23. Boyd S, Chua LO (1985) Fading memory and the problem of approximating nonlinear operators with volterra series. IEEE Trans. on Circuits and Systems, pp 1150–1161
24. Schölkopf B, Smola A, Müller KR (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput 10(5):1299–1319
25. Thomson A, West D, Wang Y, Bannister A (2002) Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2–5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro. Cerebral Cortex 12(9):936–953
26. Verma V, Simmons R, Gordon G, Thrun S (2004) Particle filters for fault diagnosis. IEEE Robotics and Automation Magazine 11(2):56–66

Harald Burgsteiner graduated from Salzburg Technical High School in the field of Electronics and Information Technology and went on to receive his M.Sc. and Ph.D. from the Graz University of Technology. He passed the exams with distinction and received his degree with honors. Mr. Burgsteiner worked as a research and teaching assistant at Prof. Maass' Institute for Theoretical Computer Science at the Graz University of Technology. His main working area was to explore new learning algorithms for neural networks on robots in real-world environments. He left the group in Spring 2003. Harald Burgsteiner is currently working at the Graz University of Applied Sciences as a Professor for Medical Informatics.

Mark Kröll is a Master student at the Institute for Theoretical Computer Science, Graz University of Technoloy. Currently he works at the Division of Knowledge Discovery, Know-Center Graz. His scientific interests are in the fields of Machine Learning and Kernel Methods.

Alexander Leopold received his B.Sc. degree in Telematics from Graz University of Technology in 2005 and is currently writing his master thesis at the Signal Processing and Speech Communication Laboratory. His research interests are computational intelligence and stochastic signal processing.

Gerald Steinbauer received a M.Sc. in Computer Engineering (Telematik) in 2001 from Graz University of Technology. He is currently researcher at the Institute for Software Technology at the Graz University of Technology and works on his Ph.D.-thesis focused on intelligent robust control of autonomous mobile robots. His research interests include autonomous mobile robots, sensor fusion, world modeling, robust robot control and RoboCup. He built up the RoboCup Middle-Size League Team of Graz University of Technology and works currently as its project leader. He is a member of the IEEE Robotics and Automation Society, the IEEE Computer Society and the Austrian Society for Artificial Intelligence. Moreover, he is co-founder and member of the Austrian RoboCup National Chapter.