Echo State Networks for Mobile Robot Modeling and Control

Conference Paper · July 2003

DOI: 10.1007/978-3-540-25940-4_14 · Source: DBLP

Paul Gerhard Plöger
Hochschule Bonn-Rhein-Sieg
79 PUBLICATIONS **475** CITATIONS

Echo State Networks for Mobile Robot Modeling and Control

Paul G. Plöger, Adriana Arghir, Tobias Günther, and Ramin Hosseiny

FHG Institute of Autonomous Intelligent Systems
Schloss Birlinghoven
53754 St. Augustin, Germany
{paul-gerhard.ploeger,adriana.arghir,tobias.guenther,
ramin.hosseiny}@ais.fraunhofer.de

Abstract. Applications of recurrent neural networks (RNNs) tend to be rare because training is difficult. A recent theoretical breakthrough [Jae01b] called Echo State Networks (ESNs) has made RNN training easy and fast and makes RNNs a versatile tool for many problems. The key idea is training the output weights *only* of an otherwise topologically unrestricted but contractive network. After outlining the mathematical basics, we apply ESNs to two examples namely to the generation of a dynamical model for a differential drive robot using supervised learning and secondly to the training of a respective motor controller.

1 Introduction

Neural networks can serve as universal dynamical system representations, thus they constitute very powerful way of modeling [MNM02]. Simultaneously they are versatile in the tasks they can solve and without doubt neuronal networks represent an extremely successful biologically inspired solution concept. Consequently researchers begin to recast technical problems in ways amenable for solving them by the help of neural networks. Topics cover system modeling, non-linear control, pattern classification or anticipation and prediction. Examples are found in form of feed-forward networks i.e. multi-layer perceptrons which allow autonomous driving of cars [Pom93] or the silicon retinas of Carver Mead [Mea89] which produce instantaneous optical flow very similar to natural processes found in the visual perception of animals. When it comes to even more interesting recurrent neural networks (RNN), users face some major problems. They find that using RNNs is in principle possible but mostly too difficult to be really applicable. Main problems are:

1. What is the 'right' structure for a RNN: i.e. which topology fits to the given problem best?
2. The convergence of teaching: i.e. which method will converge fast enough? There is a very pronounced desire for efficiency of training.
3. Over-fitting and exactness: i.e how to avoid too literal reproduction yet assure convergent behavior with respect to the teacher signal?

There is a proliferation of different approaches for point 1. e.g. Ellman nets, Jordan nets or Hopfield RNNS to name a few. Yet item 2. severely hinders to apply RNNs to larger class of problems. Known supervised training techniques comprise Back Propagation Through Time (BPTT), Real Time Recurrent Learning (RTRL) or Extended Kalman Filtering (EKF) all of which have some major drawbacks. Application of BPTT to RNNs requires stacking identical copies of the network thus unfolding the cyclic paths in the synaptic connections. Unlike back-propagation used in feed-forward nets, BPTT is not guaranteed to converge to a local error minimum, computational cost is $O(TN^2)$ per time step where N is the number of nodes, T the number of epochs [BSF94]. In contrast RTRL needs $O((N + L)^4)$ (L denotes number of output units), which makes this algorithm only applicable for small nets. The algorithm complexity of EKF is $O(LN^2)$. EKF is mathematically very elaborate and only a few experts have trained predefined dynamical system behaviors successfully [SV98]. In this article we approach items 1. and 2. from a different point of view and give a stunningly simple solution. We introduce the notion of Echo State Networks (ESNs) and apply this concept successfully in two problem domains, namely nonlinear system identification and nonlinear control. At this time, it seems that ESNs are also applicable to many others of the problems generally known to be solvable by RNNs such as filtering sensor data streams [Hou03] or classification of multiple sensory inputs [Sch02]. Thus they can be applied to many other every day problems of roboticists and their use is in no way restricted to the covered examples. See [Jae01b], [Jae01c], [Jae02] for an in-depth coverage of already investigated examples.

This article makes the following contribution to ESN related research: for the first time we successfully apply this technique to system modeling and controller generation for mobile robots. Using well known error norms from control theory we demonstrate that ESN based well-trained controller can compete with and even outperforms a classical handwritten one.

The remainder of this article is structured as follows: in section 2 we define basic notation and mathematics of ESNs. In the core part section 3 we describe in depth the process of teacher signal generation, training of system model and motor controller and give results on the soundness of the application of ESNs in the chosen application scenario. We close with a summary and give references.

2 Recurrent Neural Networks and the Echo State Property

In general, a discrete time recurrent neural network can be described as a graph with three sets of nodes, namely K input nodes $\mathbf{u}$, N internal network nodes $\mathbf{x}$ and L output nodes $\mathbf{y}$. We use the terms nodes, units and neurons interchangeably. Activation vectors at time point n are denoted by $\mathbf{u}(n) = (u_1(n), \ldots, u_K(n))$, $\mathbf{x}(n) = (x_1(n), \ldots, x_N(n))$ and $\mathbf{y}(n) = (y_1(n), \ldots, y_L(n))$ respectively. The interconnect edges are represented by weights $w_{ij} \in I\!R$, which are collected in adjacency matrices, such that $w_{ij} \neq 0$ implies there is an edge from node $j \to i$. We define $\mathbf{W}^{in}_{N\times K} = (w^{in}_{ij})$ for the input weights, $\mathbf{W}_{N\times N} = (w_{ij})$ for the internal connections, $\mathbf{W}^{out}_{L\times(K+N+L)} = (w^{out}_{ij})$ for the output weights and finally $\mathbf{W}^{back}_{N\times L} = (w^{back}_{ij})$ for the weights which project back from output nodes into the net, subscripts denote dimensions. Observe that direct impact from an input node to an output node or from one output node to another output node is possible. Evolution of the internal activation vector given by

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)) \tag{1}$$

where $\mathbf{f} = (f_1, \ldots, f_N)$ are the internal activation functions. Calculation of the new internal node vector from the current inputs, given old activation and old output according to equation (1) is called *evaluation*. The neural network computes its output activations according to

$$\mathbf{y}(n+1) = \mathbf{f}^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n)) \tag{2}$$

where $\mathbf{f^{out}} = (f^{out}_1, \ldots, f^{out}_L)$ are the output activation functions and $(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n))$ denotes concatenation of input, internal and previous output activation vectors. Equation (2) is called *exploitation*. Observe that we do not *require* recurrent pathways between internal units, although we expect them to exist, and that there is *no* restriction on the topology. In our case of echo state networks we usually have full matrices for $\mathbf{W}^{in}, \mathbf{W}^{out}$ and if needed at all also for $\mathbf{W}^{back}$. $\mathbf{W}$ is a sparse matrixes with typical densities ranging from 5-20%. Successive evaluation and exploitation of the net according to equations (1), (2) might show a chaotic, unbounded behavior. Thus it is necessary to damp the system. This can be achieved by a proper global scaling of $\mathbf{W}$ (see below). With the given notation the Echo State Property (ESP) can be stated as follows [Jae02].

Definition 1. *Assume that an RNN with* $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$ *defined like above is driven by a predefined teacher input signal* $\mathbf{u}(n)$ *and teacher-forced by an expected teacher output signal* $\mathbf{y}(n)$ *both contained in compact intervals* U^K *and* Y^L *respectively. Then the network* $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$ *has the* echo state property (ESP) *with respect to* U^K *and* Y^L *iff for every left infinite sequence* $(\mathbf{u}(n), \mathbf{y}(n-1)), n = \ldots, -2, -1, 0$ *and all state sequences* $\mathbf{x}(n), \mathbf{x}'(n)$ *which are generated according to*

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)) \tag{3}$$

$$\mathbf{x}'(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}'(n) + \mathbf{W}^{back}\mathbf{y}(n)) \tag{4}$$

it holds true that $x(n) = x'(n)$ *for all* $n \le 0$

Intuitively this means that if the network has been running long enough, its internal state is uniquely determined by the history of the input signal and the teacher forced output (see [Jae02] for details). The ESP is connected to certain algebraic properties of the weight matrix $\mathbf{W}$. There are sufficient conditions for RNNs to either proof or to disproof ESP. Since it eases the further presentation and the results do not depend on it, we assume $f_i(x), f^{out}_i = tanh(x)$ from now on.

Theorem 1. *Define σ_{max} as largest singular value of $\mathbf{W}$, λ_{max} as largest absolute eigenvalue of $\mathbf{W}$.*

(a) *If $\sigma_{max} < 1$ then ESP holds for the network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$.*
(b) *If $|\lambda_{max}| > 1$ then the network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$ has no echo states for any input/output interval $U^K \times Y^L$ which contains the zero input/output tuple $(\mathbf{0}, \mathbf{0})$.*

In practical experiments it was found that condition (a) is much too strong and that on the contrary the negation of (b) appeared to be sufficient in the sense that it always produced RNNs with ESP though this is not provable up to now. More precisely we apply the following algorithm to produce RNNs with ESP:

Algorithm 1 *1. Randomly generate a sparse matrix $\mathbf{W}_0, w^0_{ij} \in [-1, 1]$ with a low density (e.g. 5-20% weights $\neq 0$).*
2. Normalize $\mathbf{W}_1 = 1/|\lambda_{max}|\mathbf{W}_0$, where λ_{max} is eigenvalue of $\mathbf{W}_0$ with maximal absolute value.
3. Scale $\mathbf{W} = \alpha \mathbf{W}_1$, where $\alpha < 1$, so α is the spectral radius of $\mathbf{W}$.
4. Then ESP holds for the network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$

Observe that the ESP prevails regardless of the choice of ($\mathbf{W}^{in}$ or $\mathbf{W}^{back}$). Thus the open parameters of the network which require tuning are the dimensionality N of $\mathbf{W}$, spectral radius α, and scaling, sign and topology of input weights $\mathbf{W}^{in}$ and back-propagation weights $\mathbf{W}^{back}$, all of which have be adapted to the time series data of the given problem domain. The parameter α can be interpreted as the intrinsic time scale of the ESN where small α means fast reacting network and α close to one implies slow reactions. The number of inner nodes, N, relates to the short term memory capacity of the net [Jae01c]. Algorithm 1 gives a surprising answer to problem 1 from section 1: the exact interconnect topology of the RNN can be arbitrary and a condition taming the largest eigenvalues will suffice. As another convenient consequence teaching of echo state networks becomes easy and user friendly. Specifically for RNNs with ESP, *only* the matrix of output weights $\mathbf{W}^{\mathbf{out}}$ needs to be adjusted. In detail, one applies the following steps:

Algorithm 2 *Let $D = \{d_n | n = 1, \ldots, T\}$ be a set of T elements of training data d_n each consisting of a teach input vector $\mathbf{u}_{teach}(n)$ and a desired (to be taught) output vector $\mathbf{y}_{teach}(n)$. Set $\mathbf{x}(0) = \mathbf{0}$ and $\mathbf{y}_{teach}(0) = \mathbf{0}$.*

1. Calculate the current network state $\mathbf{x}(n+1), i = 0, \ldots, T_0 - 1$ according to equation (1).
2. For $n = T_0, \ldots, T$ concatenate $(\mathbf{u}^T_{teach}(n+1), \mathbf{x}^T(n+1), \mathbf{y}^T_{teach}(n))$ in rows and store it in a state collecting matrix $\mathbf{M}_{(T-T_0+1)\times(K+N+L)}$
3. Similarly collect $(\tanh^{-1} \mathbf{y}^T_{teach}(n))$ in rows into the teacher collection matrix $\mathbf{C}_{(T-T_0+1)\times L}$
4. solve for $\mathbf{W}' = \mathbf{M}^{-1}\mathbf{C}$ where $\mathbf{M}^{-1}$ denotes the pseudo (Moore-Penrose) inverse of $\mathbf{M}$ and set $\mathbf{W}^{\mathbf{out}} = \mathbf{W}'^T$.

Usually $T - T_0 + 1 >> (K + N + L)$ so step 4 amounts to the solution of an over-determined set of equations by regression, which can be accomplished by virtually any numerical SW packages in little cpu time. Usually there is no unique solution but there is unique one shortest in length. The trained network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{out}, \mathbf{W}^{back})$ is now ready to use by application of equations (1)+(2). In cases necessary arising stability problems can be cured by adding a vector term of small white noise in equation (1) during step 3. Algorithm 2 addresses item 2 - the difficult teaching problem- from section 1. ESNs teach the $L \times (K + N + L)$ coefficients of $\mathbf{W}^{out}$ only instead of the whole topology, all other parts remain untouched.

Now the mathematical basics of ESN can be summarized as follows: we can heuristically characterize them as RNNs with sparse internal interconnect topology and some restriction on the size of its maximal singular (or most of the time: eigen) value. In ESNs, only output weights are trained. Thus they ease topology and teaching related problems of classical RNNs. They have a low algorithmic complexity, allow fast teaching and are highly adaptable to AI tasks like filtering and classification. To train them, a designer still needs to fix some parameters like dimension, density and spectral radius. Signals have to be properly filtered, scaled and offset. For each of these operations there exist single or combined heuristics for an educated guess of the initial values. The most time consuming part deals with scaling of input and output ranges. Here one needs to find parameter sets specially adapted to the given problem. Speaking in terms of electrical circuit analysis shifting and scaling input/output data amounts to fixing an operating point of the ESN. Like for many other RNN models stability, data over fitting or lack of generalization abilities remains an issue also for ESNs. In the next chapter we apply ESNs to real world data stored during a game to find system models and nonlinear controllers for mobile robots.

3 Applications of ESNs to Control

Our mid size league robots have a standard differential drive with passive caster-type front wheels, so only nonholonomic movement is possible. As such the dynamics of the robot forms a nonlinear system which requires expert knowledge to be modeled in an analytic way. Consequently we preferred a black-box approach to modeling based on RNNs. They are especially powerful when approximating fast changing dynamics which is frequently the case in our behavior programming approach called dual dynamics (DD). In DD different behaviors run simultaneously on each robot [BK01].

Schematically Figure 1 displays the data flow in our robot architecture. Required left and right wheel speeds are calculated by the DD behavior program which runs on a LINUX notebook. The two values are send to the PIDs approximately every 33 msec. The PIDs convert the required speed (cm/s) into a pulse width modulation signal (PWM) in percent thus effectively controlling the voltage of the motors. At the two motors actual odometric speed values are measured. The velocity (cm/s) is feedback to the PID to close the low level control

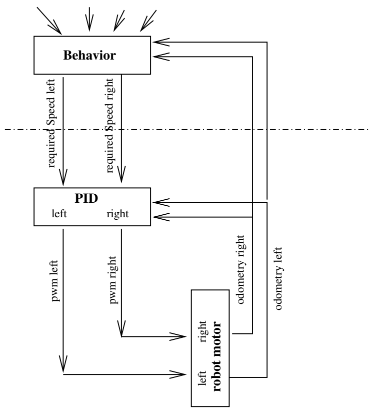

Fig. 1. Interface to low level robot architecture. Above the dashed line, we have the non-real-time PC-level. Below it there is a closed fast reactive real-time loop on μ-controller-level. Here physical modeling is mandatory.

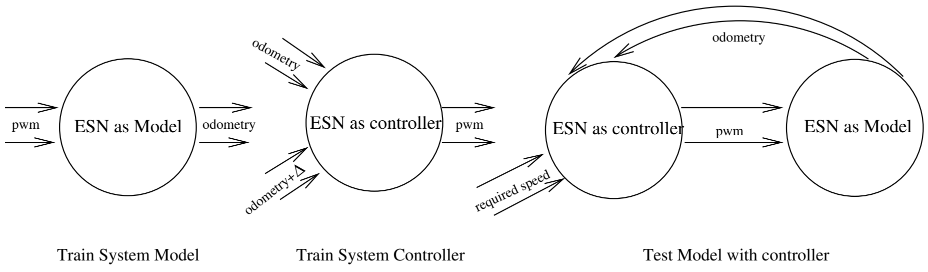

Fig. 2. Left: training of system model, middle: training of controller, right: combined simulation of controller and model, acting as a model of the physical robot.

loop and also to the behavior system running on the notebook. It operates in a non real-time way while below the dotted line the PID micro-controller operates in real-time in its feedback loop. We employ ESNs in three different situations, firstly as a system model. Inputs of the model are two PWMs and outputs are odometric velocities left and right. This can be seen as a forward model as it maps from inputs to outputs, in a context of a given state vector [Jor95]. In terms of Figure 1 this amounts to replacing the bottom box by an ESN. Secondly, as a system controller or inverse model which inverts the system by determining (i.e. by learning) the motor commands which will cause a desired modification in state. Here the trained ESN acts as controller, as it provides the necessary motor command (PWM) to achieve some desired state (i.e. the required speed). This is equivalent to the substitution of the box PID with an ESN in Figure 1. We train both replacing networks separately. Lastly in a third setup both ESNs are coupled to build an integrated model robot/low level controller pair. Then in Figure 1 the whole system below the dashed line is being modeled.

Model: Figure 2 displays the two different teaching situations and the application situation in the simulator. We begin by training the system model like depicted on the left side. According to Algorithm 1 we construct an ESN of dimension $N = 100$, 5% interconnection arcs spectral radius $\lambda = 0.8$, with two inputs and outputs (i.e. $K = 2$, $L = 2$) for left and right wheel. The inputs of the model are PWMs. Outputs are odometry and the teacher signal is set to the measured odometry from a stored trace file. The inputs were scaled down from their original domain $[-250, 250]$ to the range $[-1, 1]$. We added some mild noise of $+/-0.002$ during teaching in equation 1 as a fourth term inside the network activation function. The matrix $\mathbf{W}^{back}$ was set to zero. This is done for passive filtering tasks. In tasks involving active signal generation, the back weights are usually different from zero. The parameters and results from training are summarized in Table 1. We also computed $MSEr$ and $MSEl$ which denote the mean square error for both learned time series for the left and right wheel. Teaching took just less then a minute for a training sequence of length 6800 time steps. The evaluation time took a second on a Pentium III class machine using a MATLAB 5.30 implementation.

Table 1. Parameters used for training ESNs as Model and as Controller.

Name	**Model**	**New Contrl.**
Dimension	100	100
Density	5%	6%
λ	0.8	0.8
Inputs	2	4
Outputs	2	2
$\|w_{ij}^{in}\|$	±0.25	0.5
Noise	±0.002	±0.002
MSE left	$6.5e-5$	$4e-6$
MSE right	$8.4e-5$	$4e-6$
Train steps	6800	1800

A picture of the desired and trained time series is shown in picture Figure 3. Firstly it can be stated that the trained model follows the trainer signal quite closely in general. Taking the maximum norm the overall relative error is 12%. The L^1 integral norm of the difference function between teacher and network output defined as $\int_a^b |f-g|dt/(b-a)$ is 1.1e-2 on the given time interval, the respective L^2 error norm is 4.2e-4. Taking a closer look the following observations can be made. Firstly we see how the measurement noise on top of the teacher signal is filtered away. The ESN generated signal appears to be smoother then the orignal. Secondly in the start interval [6900,7000] the ESN signal saturates and is not able to reach the desired 175 cm/sec. The explanation for this is very simple, since the used data file contained only 212 data points above 150 cm/sec and just 10 over 180cm/sec. This data set is far too small to train the network sufficiently well in this region of high speeds. By itself an ESN can neither extrapolate nor generalize for learned situations to close nearby input stimuli yet lying beyond the previously trained range. Thus it saturates at 150cm/sec. The very same explanation applies to some observed over-drives when the robot moves back. During teaching this situation prevailed for just about 10% of the time. We do not have an convincing explanation for the divergence in intervals [7250,7350] or [7750,7800] though. It might simply be mentioned that all our observations indicate that ESNs seems to behave especially well in spiking situations while in steady state situations a drift is frequently observable. The entire system model is reasonably exact to be useful in simulations and it surpasses the kinematical model in prediction quality by far. We then compared this result to a standard approach applied in the System Identification Toolbox in MATLAB. This SW supports many different methods but the default is the *prediction error method* (PEM) which is used when no special model was given by the user. Observe that PEM is still a parametric method but it chooses its parametric model all on its own using some clever heuristics. The comparison of ESN to PEM in Figure 4 proves that both models suffer from the same flaws. The training data set contained signals with extraordinary high frequencies. Consequently the fit at all rapid changes of inputs is very good, but the low frequency part is too rarely present in the teacher signal. Thus it can be concluded that the teacher stimulus set is not rich enough. More inputs sets containing rides on a straight line are needed. A final remark on the phase difference at the very end of the test data set may be noted. It is due the numerical roundoff error in the numerical calculation of the step size which PEM must use.

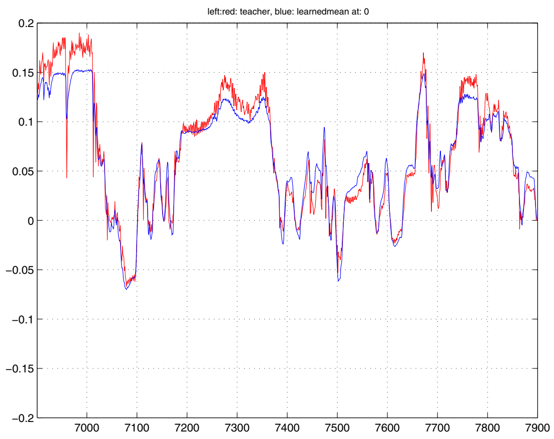

Fig. 3. Comparison of outputs of trained system model and observed robot speeds after teaching. Teacher is dashed, system is solid.

Controller: A similar teaching setup can be used to try to train a new ESN as a better controller then the given PID. If an embedded version of an ESN would be at hand this version could actually be used in the real robot replacing the

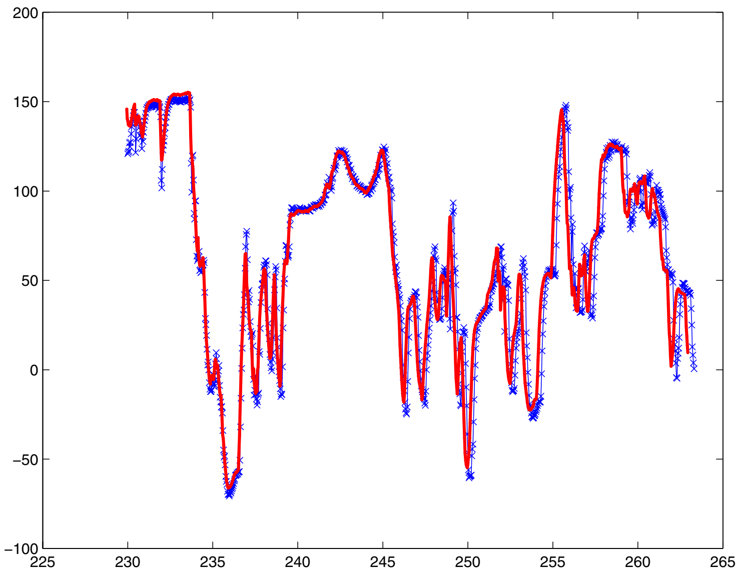

Fig. 4. Comparison of ESN method (solid line) and prediction method error (crossed line).

old PID controller. Since this hardware unit is not ready at this time we can only present a feasibility study. The ESN for this enhanced version controller has again the dimension of $N = 100$ internal units, 0.06 density, spectral radius lambda $= 0.82$, inputs $K = 4$ and outputs $L = 2$. In this case we are feeding the controller with the odometry signal itself and an incrementally delayed odometry signal (4 steps). Both are again derived from original speed data. A bigger delay it likely to enhance prediction, but would also result in damping or attenuating, which is unwanted here. In this training situation the ESN will learn to deduce how to mimic the given PWMs by using the current velocity and the desired velocity in near future. In Figure (5) we see a good fit in steady state situations on interval [50,100] as well as at rapid changes in [205,220].

Controller with System Model in the Loop: The third and last step consists of testing the new trained controller, but this time in combination the system model instead of the physical robot, see Figure 2 right frame. After initialization of start values for odometry, and PWMs, we need only to apply required speed as reference signal to the controller, while updating controller and model in a closed loop. These speeds are exactly the outputs from our DD behavior systems. Figure 5 shows the robot PID controller in the top frame and in the bottom frame it displays desired speeds, modeled odometry and PWMs. The lower part uses the new enhanced controller in combination with the system model. As it is easier, we discuss only the left motor. In the original trace on top we see a problem situation at time interval [380,460]. The desired speed is a constant but the PWM signal

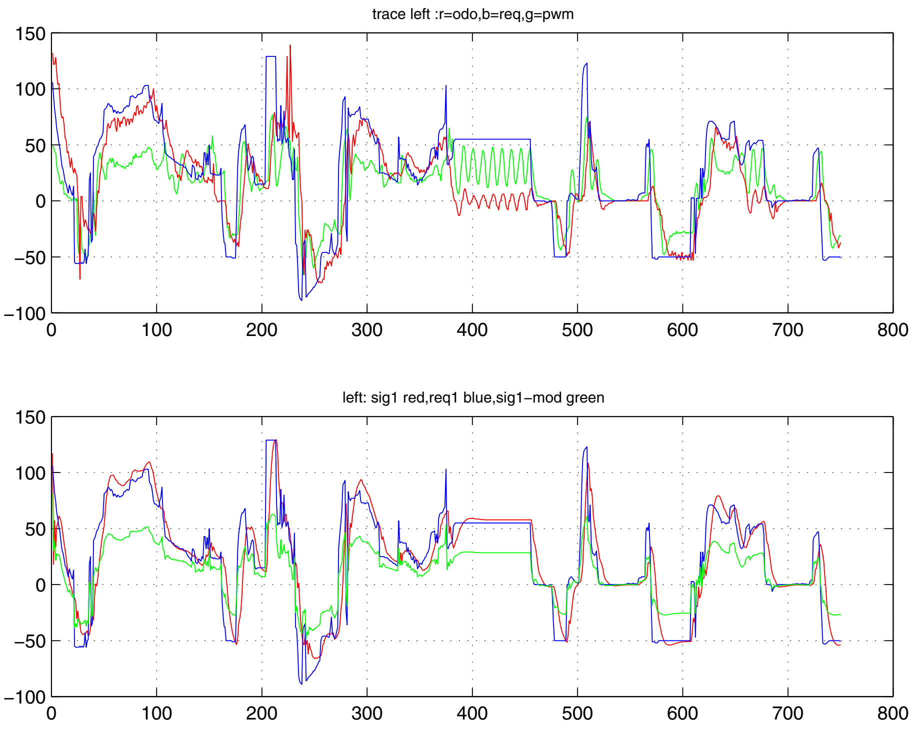

Fig. 5. Top: original trace data from real robot (black: required speed left, light grey: PWM, grey: odometry). Bottom: new enhanced controller in combination with learned physical model for an interval of 750 time steps. There are improvements at steady state situations and at points with rapid changes.

oscillates around 25 as does the measured odometry around zero. In this situation the robot was blocked by an obstacle and could not move forward as commanded by the behavior. Naturally this is an outer force not present in the simulation of the bottom frame. Instead the simulated velocity smoothly approaches the limit velocity and saturates with a significant steady state error. The situation itself was not mimicked by the trained controller instead he is able to handle it as expected with good results. This indicates that our model is well fitted. Besides that convergence in all dynamically changing situation seems to be better especially at [50,110] or at [620,660]. Another difficult situation is pictured in the time interval [200, 250]. Near step 200, required speeds are 140, but the odometry takes on this value only in the time step 225 very unstably jumping back and forth. Different from this, the bottom picture displays rectificated results for the discussed time intervals. Also the enhanced new controller can anticipate some step in future. This can be seen at time step 225. Again we computed the L^1 norm, which is one of the most popular ones in controller design, when there are fitting problems, or -as in our case- when there are big errors or "wild" points.

For the data in the original trace file the L^1 error was 2.3084 over 750 time steps. The L^1 norm for the same interval, with the optimized controller was about 0.0036.

These results clearly demonstrate the potential of our approach. Yet ESNs do share problems with other RNN based modeling approaches. For example they have to be taken as an undividable whole. This means that we can only control global network parameters like size, spectral radius etc. Changing them will impact the whole net and optimization space seems to be highly discontinuous as is it also well known from other areas like integer linear programming. Up to now we lack a systematic way to enhance convergence at one point while not sacrificing quality at others. A possible remedy might be a learnt superposition of network each being an expert in its own regime. Thus a modeling task could be decomposed and fine tuned in different independent areas. The harder or "wilder" you train the model, the better your controller will work. Another point, which is well known from learning theory, is that ESNs cannot master situations which have never been taught. The system model has to be taught "beyond limits" to be really applicable in the whole needed dynamical range. Thus in a future training sequence we want to expose the system model to dynamically wider situation by training via human operated joystick control with higher gains in comparison to the gains which the final running robot will have.

4 Summary

We introduced the notion of Echo State Networks as an easily trainable versatile variant of recurrent neural networks. We showed how these networks can be used to teach a physical system model of a differential drive robot and also how to mimic a given controller for it. Furthermore an improved controller was generated. All three applications show good agreement with observed real life data. Now an ESN can be implemented in the actual HW of the motor controller, yet the performance of this extended approach has to be studied in a future paper.

References

[A00] Christaller Th. Jaeger H Kobialka H.-U Schoell P Bredenfeld A. Robot behavior design using dual dynamics. *Tech. GMD Report*, 2000.

[Ark98] R. C. Arkin. *Behavior-based robotics.* The MIT Press, 1998.

[BK01] Ansgar Bredenfeld and Hans-Ulrich Kobialka. Team cooperation using dual dynamics. In Markus Hannebauer, editor, *Balancing reactivity and social deliberation in multi-agent systems*, Lecture notes in computer science, pages 111 – 124, 2001.

[BSF94] Yoshua Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. (Special Issue Dynamic and Recurrent Neural Networks.).

[Gal80] C. R. Gallistel. *The Organization of Action: a New Synthesis.* Lawrence Erlbaum Associates, Inc., Hilldale, NJ., 1980.

[Hou03] Ramin Housseiny. Echo state networks used for the classification and filtering of silicon retina signals. Master's thesis, RWTH AAchen, 2003.

[Jae01a] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. *GMD Report 148*, pages 1–43, 2001.

[Jae01b] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks. Technical report, GMD - Forschungszentrum Informationstechnik GmbH, 2001.

[Jae01c] Herbert Jaeger. Short term memory in echo state networks. Technical report, GMD Forschungszentrum Informationstechnik GmbH, 2001.

[Jae02] Herbert Jaeger. Tutorial on trainig recurrent neural networks covering bppt, rtrl, ekf and the "echo state network" approach. Technical report, GMD Forschungszentrum Informationstechnik GmbH, 2002.

[JD92] M.I. Jordan and Rumelhart D.E. Forward models: Supervised learning with a distal teacher, 1992.

[Jor95] M.I. Jordan. Computational aspects of motor control and motor learning. In H. Heuer and S. Keele, editors, *Handbook of Perception and Action: Motor Skills.* Academic Press, New York, 1995.

[KBM98] D. Kortenkamp, R. P. Bonasso, and R. Murphy. *Artificial Intelligence and Mobile Robots.* AAAI Press / The MIT Press, 1998.

[KS87] Furawaka K. Kawato, M. and R. Suzuki. A hierarchical neural network model for the control learning of voluntary movements, 1987.

[Mea89] Carver Mead. *Analog VLSI and Neural Systems.* Addison-Wesley, Reading, MA, USA, 1989.

[MNM02] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 2002.

[MW96] R.C. Miall and D.M. Wolpert. Forward models for phisiological motor control, 1996.

[Pom93] Dean A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* Kluwer, Dordrecht, The Netherlands, 1993.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, 2 edition, 1992.

[SB81] R.S. Sutton and A.G. Barto. Toward a modern theory of adaptive networks: expectation and prediction., 1981.

[Sch02] Frank Schoenherr. Learning to ground fact symbols in behavior-based robots. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 708–712. ECAI, IOS Press, 2002.

[SV98] Johan A.K. Suykens and Joos Vandewalle, editors. *Nonlinear modeling*, chapter Enhanced Multi-Stream Kalman Filter Training for Recurrent Networks. Kluwer, 1998.