Event Detection and Localization in Mobile Robot Navigation Using Reservoir Computing

CONFERENCE PAPER · JANUARY 2007

DOI: 10.1007/978-3-540-74695-9_68 · Source: DBLP

5 AUTHORS, INCLUDING:

Eric Antonelo
Federal University of Santa Catarina
21 PUBLICATIONS **160** CITATIONS
SEE PROFILE

Dirk Stroobandt
Ghent University
179 PUBLICATIONS **1,706** CITATIONS
SEE PROFILE

Event detection and localization in mobile robot navigation using reservoir computing

Eric A. Antonelo[1], Benjamin Schrauwen[1], Xavier Dutoit[2], Dirk Stroobandt[1], Marnix Nuttin[2]

[1] Electronics and Information Systems Department, Ghent University, Belgium
[2] Department of Mechanical Engineering - Katholic University of Leuven, Belgium
Eric.Antonelo@elis.ugent.be

Abstract. Reservoir Computing (RC) uses a randomly created recurrent neural network where only a linear readout layer is trained. In this work, RC is used for detecting complex events in autonomous robot navigation. This can be extended to robot localization based solely on sensory information. The robot thus builds an implicit map of the environment without the use of odometry data. These techniques are demonstrated in simulation on several complex and even dynamic environments.

1 Introduction

Autonomous robot navigation systems have been extensively developed in the literature [1–4]. Early navigation strategies are either deliberative (generation of robot trajectories based on path planning) or reactive (robot control based on a direct mapping of sensory input to actions). Current state-of-the-art autonomous robot control architectures are *hybrid* [1]: they have an underlying reactive controller which takes care of the real-time critical and simple tasks such as obstacle avoidance; while an upper deliberative control layer steers this reactive part. Information flow in this architecture is both downwards, from abstract deliberative tasks to concrete physical reactive behaviours, and upwards, from physical data to abstract symbols used for deliberation.

This paper investigates two cases of upward information flow: a system for recognizing complex robot events in particular environments (such as detecting if the robot goes through a door, given only sensory input); and a system for determining the current robot location, solely based on sensory information. Both are achieved using the same setup.

These tasks have been shown to be difficult [5]. Traditional algorithms based on the Simultaneous Localization and Mapping (SLAM) concept are expensive to implement due to limited computational efficiency and also hold uncertainties during the calculation of the robot's pose [5].

This work uses an implicit way of forming a representation of the robot's environment that is based on a Recurrent Neural Network (RNN), more specifically using Reservoir Computing (RC). This is a term that groups three similar computing techniques, namely, Echo State Networks [6], Liquid State Machines [7],

and BackPropagation DeCorrelation [8]. All three techniques are characterized by having an RNN that is used as a reservoir of rich dynamics and a linear readout output layer. Only the readout layer is trained by supervised learning, while the recurrent part of the network (the so called reservoir) has fixed weights, but is scaled so that its dynamic regime is at the edge of chaos. Theoretical analysis of reservoir computing methods [9] and a broad range of applications [10] (which often even drastically outperform the current state-of-the-art [11]) show that RC is very powerful and overcomes the problems of tradititonal RNN training such as slow convergence and computational requirements.

The short-term memory, present in these networks, is crucial for solving the event detection and localization tasks. It is not only the instantaneous sensory inputs that are needed to solve the tasks, but also the sensory history [12] and dynamics.

It has already been shown in [13] that RC can be used to detect events in an autonomous robot setting. This work extends these results by also considering dynamic environments for event detection, and goes largely beyond that work by using it to construct implicit maps of the environment for robot localization.

The idea of employing a neural network as a localization model for the robot is also inspired by biological systems. Experiments accomplished with rats show that the hippocampus in their brain forms activation patterns that are associated with locations visited by the rat. These so called *place cells* are the most common evidence for such fact. They fire when an animal is in a particular location in its environment [14].

The data (robot sensors and actuators) are generated using a simulator developed in [3]. It is a completely reactive controller trained by reinforcement learning to explore the environment. The dataset collected from the simulator is used to train an RC system in order to detect events as well as to predict the robot location in particular environments.

2 Reservoir computing

The current work uses the Echo State Network approach as a learning system for detection of events as well as for robot localisation. The random, recurrent neural network (or reservoir) is composed of sigmoidal neurons and is modelled by the following state update equation:

$$\mathbf{x}(t+1) = f(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t)), \tag{1}$$

where: $\mathbf{W}_{\text{in}}$ is the connection matrix from input to reservoir; $\mathbf{W}$ is the weight matrix for the recurrent connections between internal nodes; f is the hyperbolic tangent function; and $\mathbf{u}(t)$ is the input vector at time t. The initial state is $\mathbf{x}(0) = \mathbf{0}$.

The output $\mathbf{y}(t)$ of the network at time t is given by

$$\mathbf{y}(t) = \mathbf{W}_{\text{out}} \begin{bmatrix} \mathbf{x}(t) \\ 1 \end{bmatrix}, \tag{2}$$

where $\mathbf{W}_{\text{out}}$ is the readout matrix.

The matrices $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ are fixed and randomly created at the beginning. If n_i and n_r denote the number of inputs and neurons inside the reservoir, respectively, then $\mathbf{W}_{\text{in}}$ is a $n_r \times n_i$ matrix. $\mathbf{W}$ is a $n_r \times n_r$ matrix where each element is drawn from a normal distribution with mean 0 and variance 1. It is then re-scaled by first dividing the matrix by its spectral radius (the largest absolute eigenvalue) and then multiplying it by 0.95. The spectral radius of the rescaled matrix is thus 0.95 which is close to the edge of stability (around a spectral radius 1). The value of the spectral radius could be further optimised for each experiment, but a quick grid search showed that 0.95 is near optimal for all cases.

The $(n_r + 1) \times n_o$ matrix $\mathbf{W}_{\text{out}}$ is the only matrix trained during the experiment (with n_o denoting the number of outputs). Let $\hat{\mathbf{y}}_{\text{fish}}(t)$[3] denote the desired output at time t, then the readout matrix is created by solving (in the mean square sense) the following equation:

$$\mathbf{W}_{\text{out}} \cdot \begin{bmatrix} \mathbf{x}(1) & \mathbf{x}(2) & \dots & \mathbf{x}(n_t) \\ 1 & 1 & \dots & 1 \end{bmatrix} = [\hat{\mathbf{y}}_{\text{fish}}(1)\ \hat{\mathbf{y}}_{\text{fish}}(2)\ \dots\ \hat{\mathbf{y}}_{\text{fish}}(n_t)], \tag{3}$$

where n_t is the total number of time samples.

Prior to training, the desired outputs are relabelled in order to optimise the classification results. Each line of the original target data $\hat{\mathbf{Y}}$ represents one desired output over time, and each output consists of $+1$ and -1. But the number of positive desired outputs can be fairly different from the number of negative desired outputs in each line. In order to get optimal classification through regression (i.e. through solving (3) in the least square sense), each element $\hat{y}^i(t)$ of the i-th line $\hat{\mathbf{y}}^i$ of $\hat{\mathbf{Y}}$ is rescaled so that the whole line $\hat{\mathbf{y}}^i$ sum up to 0:

$$\hat{y}^i_{\text{fish}}(t) = \begin{cases} \frac{n^i_+ + n^i_-}{n^i_+} & \text{if } \hat{y}^i(t) > 0 \\ -\frac{n^i_+ + n^i_-}{n^i_-} & \text{if } \hat{y}^i(t) < 0 \end{cases}, \tag{4}$$

where $n^i_+ = |\{\hat{y}^i(t) | \hat{y}^i(t) > 0\}|$ and $n^i_- = |\{\hat{y}^i(t) | \hat{y}^i(t) < 0\}|$ denote the number of positive and negative required outputs in the i-th line of $\hat{\mathbf{Y}}$, respectively.

3 Robot Model and Controller

The dataset used to train reservoir networks is generated by a simulator used in [3]. Next the environment and robot controller are described briefly. The environment of the robot is composed of repulsive and attractive objects. Each object has a particular color, denoting its respective class. Obstacles are considered repulsive objects while targets are attractive objects [2]. The robot model is shown in Fig. 1. The robot interacts with the environment by distance, color and

[3] $\hat{\mathbf{y}}_{\text{fish}}(t)$ refers to the desired output after applying the *fisher* labeling given by (4)

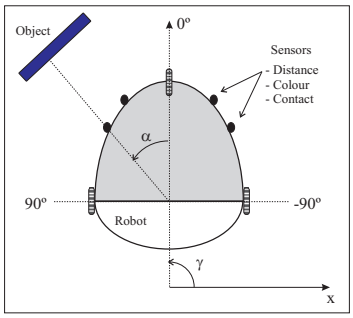


Fig. 1. Robot model

contact sensors; and by one actuator that controls the adjustment on the movement direction. Sensor positions are distributed homogenously over the front of the robot (from -90° to +90°). Each position holds three sensors (for distance, color and contact perception) [2]. In this work, the robot model has 17 sensor positions, differing from [3]. The velocity of the robot is constant. At each iteration the robot is able to execute a direction adjustment to the left or to the right in the range [0, 15] (degrees).

The robot controller (based on [3]) is composed of hierarchical neural networks which are adjusted by classical reinforcement learning mechanisms. The controller constructs its navigation strategy as the robot interacts with the environment. Only already trained robot controllers, which all show very good exploratory behavior after training, are used for generating data. The data (from distance and color sensors, and actuator) collected from the robot simulator are used to train and test reservoir networks in a Matlab environment using the RCT Toolbox[4] [10]. Gaussian noise is added to distance sensors data by the robot simulator.

4 Event Detection in Robot Navigation

Event detection in noisy environments is not a trivial task. There can be very similar scenes from the robot's perspective so that precise event detection becomes very difficult to accomplish [9]. Two different experiments are conducted for the event detection task. The environments used are shown in Fig. 2. They are composed of a large (blue) corridor with a (yellow) target at each end (they appear as dark and light gray objects in black and white format). During simulation, the robot keeps navigating through the corridor and capturing the targets (that are sequentially put back in the same location). There are four possible events of predefined duration and location, which are labeled in Fig. 2. The interpretation should be: when the robot passes through a predefined location, an

[4] This is an open-source Matlab toolbox for Reservoir Computing which is freely available at http://www.elis.ugent.be/rct

event should be detected (e.g. entering the corridor corner area, passing through the middle of the corridor). The second environment is the same as the first environment, except for a new blinking object in the middle of the corridor (with random blink interval) which can block the robot's way.

Experiment 1 is accomplished considering the first environment and experiment 2 takes place in the second environment. Both experiments take 120.000 time steps of simulation time. The original dataset is resampled by a factor of 100, resulting in a smaller dataset of 1.200 observations. The original sampling rate is high (it takes approximately 2000 time steps to go from one side of the simulation environment to the other), which is useful to efficiently control the robot with the implemented controller. But when applying the sensory input to the reservoir it is very important that the internal dynamics and memory of the reservoir are in the same temporal range as the temporal range needed to solve the task. We achieved this by resampling the input, which effectively slows down the dynamics in the reservoir [15]. A grid search of the resampling rate showed that the optimal time range was achieved with a resampling factor of 100.

The inputs to the network are distance and color sensors and a robot actuator (current direction adjustment) suming up 35 inputs which can range from 0 to 1. Parameter configuration is as follows. The reservoir is composed of 400 nodes, scaled to a spectral radius of $|\lambda_{max}| = 0.95$. The readout layer has 4 output units (one for each event detector) which are postprocessed by a winner-take-all function. This function sets the output of the most activated neuron to 1 whereas the others are set to -1. Note that if all the neurons output a negative value, then the winner-take-all function set every output to -1 (this means no event is detected). The input nodes are connected to reservoir nodes by a fraction of 0.3 and are set to -0.15 or 0.15 with equal probabilities. The performance measure considers the number of mispredicted observations and is based on a 3-fold cross-validation method (so, 400 observations, resampled from 40.000 time steps, are selected as test data).

The results are shown in Fig. 3 and summarized in Table 1. Each experiment is evaluated 30 times with different stochastically generated reservoirs and the results are averaged over these 30 runs. It is possible that the robot develops a cyclic and rhythmic trajectory in experiment 1 (see Fig. 3). The trained reservoir is able to detect the 4 events very precisely with a performance of 99 % on test data. One could argue that the reservoir learns to recognize the rhythmic behavior and not the actual event per se. Experiment 2 is devised to test this hypothesis. It shows that a reservoir can still detect the events precisely (performance of 95.8 % on unseen data) even though a dynamic object breaks the rhythmic robot trajectory.

5 Localization in Robot Navigation

The previous section has shown that an RC network can be used to detect complex events in robot navigation with rather good performance. Now this section extends the experiments to robot localization tasks. Instead of only detecting

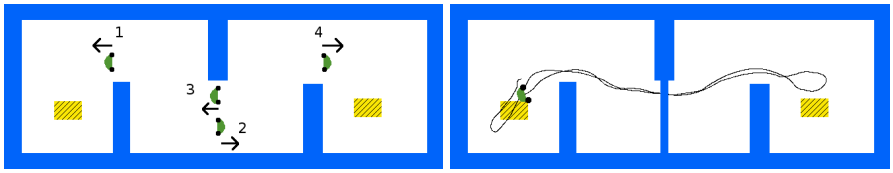


Fig. 2. Environments used for the event detection task. Four events are labeled and shown graphically (by arrows) in the first enviroment. The second environment adds a dynamic obstacle in the middle of the corridor, indicated by an arrow. A typical robot trajectory (after controller learning) is shown in the second environment. Two boxes in the environment are used as targets for the robot.

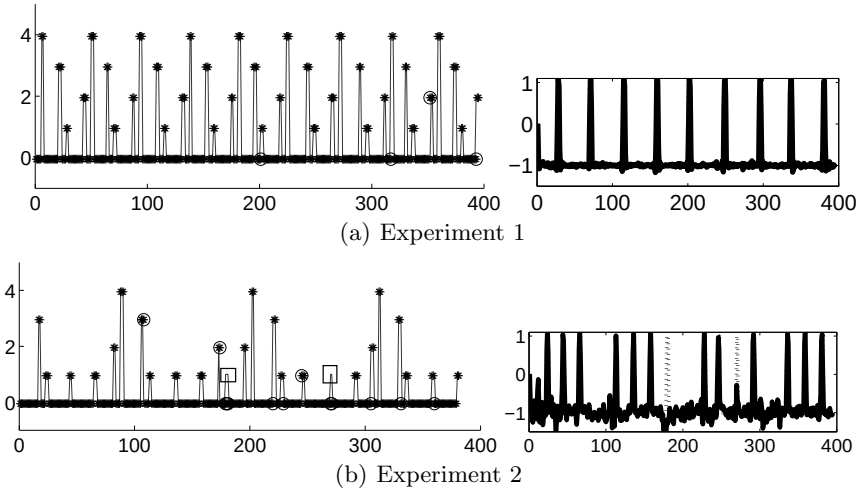


Fig. 3. Left: Events plot during a simulation for experiment 1 (a) and experiment 2 (b). An asterisk represents the predicted event by the reservoir (on test data). The actual events are points connected by lines. The mis-predictions are labeled by an extra circle. Two small rectangles emphasize two events that are not recognized by the reservoir in the bottom plot. Right: Neuron output for detecting event 1 for experiment 1 (a) and experiment 2 (b). The thick line is the actual neuron output whereas the dashed line is the desired outcome (not visible in (a)).

events, we rather want to predict the current location of the robot based on the same kind of sensory information (giving rise to a more difficult and interesting problem). Localization (or position detection) for mobile robots is usually computationally expensive in terms of space and time requirements [5]. Traditional algorithms are based on explicit maps which must be constructed before robot localization is possible. This section shows how a reservoir can be used for robot localization. Similar work which uses a Long-Short Term Memory RNN for this task is described in [16].

Two maze-like environments are used for the robot localization task (see Fig. 4). The first environment contains 64 predefined locations, that are displayed by small triangles labeled by numbers. Differently from [16], the entire environment is tagged with labels (not only rooms). This feature makes it possible to use a more precise trajectory planner.

The same reservoir parameter configuration as in the previous section is used for the following experiments. The resampling rate for the dataset is also 100. Exceptions are: the size of the readout layer is equivalent to the number of predefined locations in the environment; and the postprocessing function for the readout units is the winner-take-all function which always takes the most activated neuron and set it to 1 (the others are set to -1). So, there is always a predicted location (in contrast to the no event detected situation in previous section). Here also a 3-fold cross-validation is used for performance measure (last experiment is based on a 6-fold cross-validation).

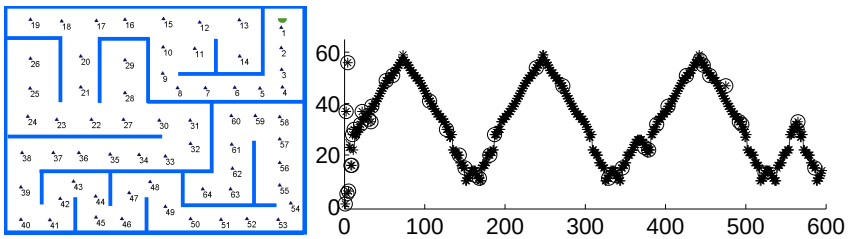
Experiment 3 is accomplished with the first environment from Fig. 4 and lasts 180.000 time steps (before resampling). The resulting robot occupancy grid can be seen in the same figure: it shows that the reservoir is predicting the robot location very well (on test data), with very few mispredictions that are not far located from the actual location (10.3 % is the test error, see Table 1).

Experiment 4, accomplished in the second environment, represents a new challenge for the reservoir-based position detector: the environment has several symmetries and identical areas. For instance, going from position 27 to 26 looks the same for the robot as going from position 22 to 24. The simulation has 120.000 time steps. The resulting occupancy grid in Fig. 4 shows an efficient position detector, featuring a performance of 87.6 % of correct predictions on test data (see Table 1).

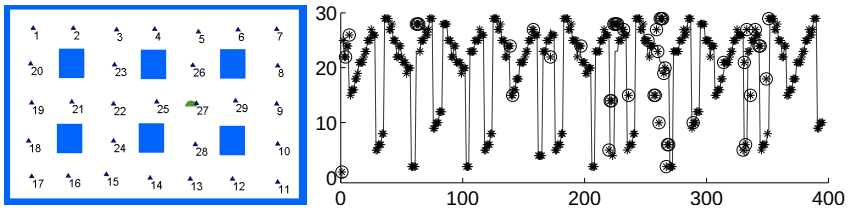
Experiment 5 uses the third environment in Fig. 4, that is the same as the first environment, but with additional 11 slow moving obstacles distributed through the environment (moving obstacles are also considered in [16]). These dynamic objects change the robot behavior and also add more noise to sensor readings. The simulation has 360.000 time steps. The respective occupancy grid in Fig. 4

Table 1. Summarized results. For each experiment, thirty (30) runs with the same robot dataset are accomplished (that is resampled by a factor of 100). Every run is based on an 3-fold cross-validation method (experiment 5 uses 6-fold cross-validation). The trainning and test errors are the mean over these 30 runs. The first two experiments are event detection tasks whereas the other three are robot localization tasks. Experiments 2 and 5 consider dynamic objects in the environment.

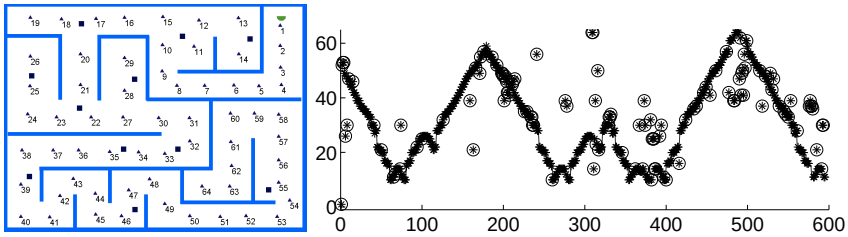
Experiment	Time steps	Train Error	Test Error
1	1200	0.6 %	1.0 %
2	1200	0.9 %	4.2 %
3	1800	2.9 %	10.3 %
4	1200	1.7 %	12.4 %
5	3600	10.9 %	22.1 %



(a) Experiment 3



(b) Experiment 4



(c) Experiment 5

Fig. 4. Environments used for the experiments (left) and respective resulting robot occupancy grids (right). First environment is tagged with 64 labels displayed by small triangles. In the occupancy grid, an asterisk represents the predicted location (on test data, that is 1/3 of the total data) while connected points are the actual robot positions. Mispredicted locations display an additional circle. The second environment has 29 labels distributed through very similar areas. The third environment is the same as the first environment but with additional slow moving obstacles (represented by small rectangles) which add more noise and dynamics to sensor readings and to the robot trajectory, respectively.

shows that the reservoir is correct in most of the predictions (77.8 %). Some of the mispredictions are located a bit further from the actual position, due to the new source of dynamics and noise.

Experiments only considering distance sensors (removing actuator and color sensor data) result in the same performance reported for the previous experiments in this section. The reservoir network also copes with the kidnapping situation (also reported in [16]). In a new experiment using the environment from experiment 3, the robot is replaced from location 54 to location 27. The

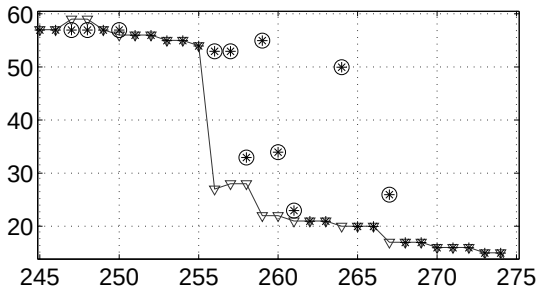


Fig. 5. Occupancy grid after kidnapping the robot in first environment of Fig. 4. An additional triangle is placed at the actual robot position. At time step 255, the robot is moved from position 54 to position 27. The reservoir network takes 7 time steps until it first predict sucessfully the current robot position (at time step 262). The robot visits 4 locations (27, 28, 22 and 21) until the sucessfull prediction.

network is able to predict sucessfully the robot position after 7 time steps (see Fig. 5). Note that the RC network is not trained with the kidnapping situation.

6 Conclusions and future work

In this work we show that it is possible to detect complex events and locate a robot in even dynamic environments with a random dynamic system which is processed by just a single linear readout layer. The proposed system shows very good performance in difficult environments such as mazes or environments which are highly symmetric. To achieve this we only use the dynamics of the sensory information, not the actual behaviors (as in [13]). Besides, no decrease on reservoir performance is reported when experiments on robot localization only consider distance sensors.

This paper only scratched the surface of what could be possible with this technology. As future work we plan to implement it on a real robotic platform, as it is considered the standard and best evaluation method for robotic systems. In this way we can also make comparisons to existing SLAM techniques. Additionaly, a deliberative robotic system can now be constructed so that actual path planning and navigation is accomplished based on the information gathered by the RC-based localization system.

From an RC view, we could improve performance by tuning the reservoir dynamics and time scales for different tasks. For instance, experiments with robots with variable speeds during simulation can be tackled by inducing distinct reservoir dynamics (creating different time scales in the reservoir operation). Future work also includes the unsupervised detection and generation of locations, much resembling actual place cells. Finally, the implicit map stored in the reservoir could be made explicit by using an RC system in a generative setting: given a location as input, the reservoir could start creating expectations (much like 'dreaming') of possible paths and environments.

Acknowledgements This research is partially funded by FWO Flanders project G.0317.05.

References

1. Arkin, R.: Behavior-based robotics. MIT Press (1998)
2. Antonelo, E.A., Figueiredo, M., Baerlvedt, A.J., Calvo, R.: Intelligent autonomous navigation for mobile robots: spatial concept acquisition and object discrimination. In: Proceedings of the 6th IEEE International Symposium on Computational Intelligence in Robotics and Automation, Helsinki, Finland (2005)
3. Antonelo, E.A., Baerlvedt, A.J., Rognvaldsson, T., Figueiredo, M.: Modular neural network and classical reinforcement learning for autonomous robot navigation: Inhibiting undesirable behaviors. In: Proceedings of IJCNN 2006, Vancouver, Canada (2006)
4. Guivant, J., Nebot, E., Baiker, S.: Autonomous navigation and map building using laser range sensors in outdoor applications. Journal of Robotics Systems **17**(10) (2000) 565–583
5. Bailey, T., Durrant-Whyte, H.: Simultaneous localisation and mapping (SLAM): Part ii state of the art. Robotics and Automation Magazine (2006)
6. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology (2001)
7. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation **14**(11) (2002) 2531–2560
8. Steil, J.J.: Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity. In: Proceedings of IJCNN '04. Volume 1. (2004) 843–848
9. Jaeger, H.: Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology (2001)
10. Verstraeten, D., Schrauwen, B., D'Haene, M., Stroobandt, D.: A unifying comparison of reservoir computing methods. Neural Networks **20** (2007) 391–403
11. Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. Science **308** (2004) 78–80
12. Schönherr, K., Cistelecan, M., Hertzberg, J., Christaller, T.: Extracting situation facts from activation value histories in behavior-based robots. In: KI-2001: Advances in Artificial Intelligence (Joint German/Austrian Conference on AI, Proceedings),, Springer (LNAI 2174) (2001) 305319
13. Hertzberg, J., Jaeger, H., Schönherr, F.: Learning to ground fact symbols in behavior-based robots. In: Proceedings of the 15th European Conference on Artificial Intelligence. (2002) 708–712
14. O'Keefe, J., Dostrovsky, J.: The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. Brain Research **34** (1971) 171–175
15. Jaeger, H., Lukosevicius, M., Popovici, D.: Optimization and applications of echo state networks with leaky integrator neurons. Neural Networks **20** (2007) 335–352
16. Forster, A., Graves, A., Schmidhuber, J.: RNN-based learning of compact maps for efficient robot localization. In: Proceedings of ESANN. (2007)