Event detection and localization for small mobile robots using reservoir computing

E. A. Antonelo [*,1], B. Schrauwen and D. Stroobandt

Department of Electronics and Information Systems, Ghent University
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

Abstract

Reservoir Computing (RC) techniques use a fixed (usually randomly created) recurrent neural network, or more generally any dynamic system, which operates at the edge of stability, where only a linear static 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 which is solely based on low-range, high-noise sensory data. The robot thus builds an implicit map of the environment. These techniques are demonstrated in both a simple simulation environment and in the physically realistic Webots simulation of the commercially available e-puck robot, using several complex and even dynamic environments.

Key words: Reservoir Computing, Robot localization, Event detection

1 Introduction

Autonomous robot navigation systems have been extensively developed in the literature [1–3]. 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 basic behaviors such as obstacle avoidance; while an upper deliberative control layer steers this reactive part for performing declarative high level tasks such as planning. Information

* Corresponding author. Tel.: +32/9 264 34 04, fax: +32/9 264 35 94
Email address: eric.antonelo@elis.ugent.be (E. A. Antonelo).
URL: http://www.elis.ugent.be/SNN (E. A. Antonelo).
[1] Eric A. Antonelo is sponsored by BOF grant 01W03907.

flow in this architecture is both downwards, from abstract deliberative tasks to concrete physical reactive behaviors, and upwards, from physical data to abstract symbols used for deliberative planning.

This paper investigates two cases of upward information flow: a system for recognizing complex events in particular environments (such as detecting if the robot goes through a door); and a system for determining the current robot location. Both are based solely on low-range, high-noise sensory information, typically found in small and cheap mobile robots. Both tasks are achieved using the same setup.

These tasks have been shown to be difficult [4]. Traditional algorithms based on the Simultaneous Localization and Mapping (SLAM) concept are expensive to implement due to high computational and memory demands and also hold uncertainties during the calculation of the robot's pose [4]. They usually need high precision ranging data from, for example, a 2D laser range scanner. These devices are currently still very expensive, consume a lot of power, and cannot be applied in small robots. Cheap, small and lightweight robots that have a high battery autonomy will thus not be able to use a SLAM based approach. These robot platform usually only have access to a limited number of ranging sensors which are low range and have high noise.

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 [5], Liquid State Machines [6], and BackPropagation DeCorrelation [7]. All three techniques are characterized by having a fixed (usually random) RNN that is used as a reservoir of rich dynamics and a linear static 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 stability. Theoretical analysis of reservoir computing methods [8] and a broad range of applications [9] (which sometimes even drastically outperform the current state-of-the-art [10]) show that RC is very powerful and overcomes many of the problems of traditional RNN training such as slow convergence, bifurcations and high 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 [11] and dynamics.

It has already been shown in [12] 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 on rats show that the hippocampus forms activation patterns that are associated with locations visited by the rat. These so called *place cells* encode the spatial location of the animal into its environment. They fire when the animal is in a particular location [13]. A similar approach is used in this work where distinct outputs are used to encode specific locations in the environment.

The experiments in this work [2] are performed using both a simple simulator developed by [2] and the physically realistic Webots [15] simulation of an e-puck robot [16]. The datasets generated by these simulators are used to train a RC system to detect events as well as to predict the robot location in several complex and dynamic environments. The training is done in a supervised fashion, but we plan to develop an autonomous and on-line way of learning novel locations as the robot drives in its environment (resembling the place cells in biological systems).

This work is organized as follows. In Section 2 we give an overview of the RC setup used in this work. Section 3 presents the two different robot models and simulators used in the experiments. The experimental results for event detection and robot localization are presented in Section 4 and 5 respectively. We conclude and present future research directions in Section 6.

2 Reservoir computing

In this work, we use the Echo State Network approach as learning model for performing event detection as well as robot localization. An ESN is composed of a discrete hyperbolic-tangent RNN (i.e., the reservoir) and a linear readout output layer which maps the reservoir states to the desired output. The general state update equation for the nodes in the reservoir and the readout output equation are as follows:

$$\mathbf{x}(t+1) = f\left(\mathbf{W}^{\text{res}}_{\text{res}}\mathbf{x}(t) + \mathbf{W}^{\text{res}}_{\text{inp}}\mathbf{u}(t) + \mathbf{W}^{\text{res}}_{\text{out}}\mathbf{y}(t) + \mathbf{W}^{\text{res}}_{\text{bias}}\right) \quad (1)$$

$$\mathbf{y}(t+1) = \mathbf{W}^{\text{out}}_{\text{res}}\mathbf{x}(t+1) + \mathbf{W}^{\text{out}}_{\text{inp}}\mathbf{u}(t) + \mathbf{W}^{\text{out}}_{\text{out}}\mathbf{y}(t) + \mathbf{W}^{\text{out}}_{\text{bias}} \quad (2)$$

where: $\mathbf{u}(t)$ denotes the input at time t; $\mathbf{x}(t)$ represents the reservoir state; $\mathbf{y}(t)$ is the output; and $f() = \tanh()$ is the hyperbolic tangent activation function. The initial state is set to $\mathbf{x}(0) = \mathbf{0}$. All weight matrices to the reservoir

[2] This paper is an extended version of [14] which was presented at ICANN 2007.

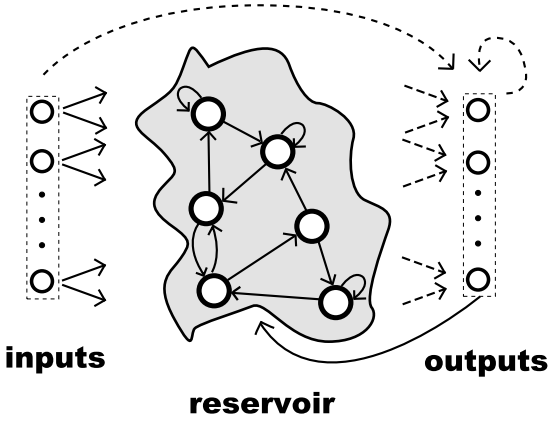


Fig. 1. Reservoir Computing network. The reservoir is a dynamical system of recurrent nodes. Solid lines represent connections which are fixed. Dashed lines are the connections which can be trained.

(denoted as $\mathbf{W}_{\cdot}^{\text{res}}$) are initialized randomly (represented by solid arrows in Fig. 1), while all connections to the output (denoted as $\mathbf{W}_{\cdot}^{\text{out}}$) are trained (represented by dashed arrows in Fig. 1).

However, for the experiments in this work, we discard the readout's output feedback to the reservoir and we add a leak rate α as in [17] to the state update equation:

$$\mathbf{x}(t+1) = f\left((1-\alpha)\mathbf{x}(t) + \alpha(\mathbf{W}_{\text{res}}^{\text{res}}\mathbf{x}(t) + \mathbf{W}_{\text{inp}}^{\text{res}}\mathbf{u}(t) + \mathbf{W}_{\text{bias}}^{\text{res}})\right). \quad (3)$$

The output calculation gets simpler once we do not use the direct connections from input to output neither the connections from output to output:

$$\mathbf{y}(t+1) = \mathbf{W}_{\text{res}}^{\text{out}}\mathbf{x}(t+1) + \mathbf{W}_{\text{bias}}^{\text{out}}. \quad (4)$$

The leak rate can effectively tune the dynamics of the reservoir. If the leak rate is chosen correctly, the reservoir dynamics can be adjusted to match the timescale of the input flow, making it possible to achieve enhanced performance (this can also be achieved by resampling the input [18,14]). In this work, some experiments use 3 pools of neurons in the reservoir with distinct leak rates to achieve better performance. Further investigation about timescales in reservoirs and leaky integrator neurons can be found in [17,18].

Each element of the connection matrix $\mathbf{W}_{\text{res}}^{\text{res}}$ is drawn from a normal distribution with mean 0 and variance 1. For most applications, the best performance is attained with a reservoir that operates at the edge of stability. The randomly created $\mathbf{W}_{\text{res}}^{\text{res}}$ matrix is rescaled such that the system is stable. This can be accomplished by rescaling the matrix so that the spectral radius (the largest absolute eigenvalue) of the linearized system is slightly smaller than one [8]. In this work we scale all reservoirs to a spectral radius of of $|\lambda_{max}| = 0.9$ which is

near optimal for most experiments, but the value of the spectral radius could be further optimized for each experiment separately.

Training is performed using either linear regression (least squares method) or ridge regression [19]. In the latter, the regularization parameter is found by grid search on a validation set. The computational efforts for training are related to computing the transpose of a matrix and matrix inversion. It takes just a few seconds to train a RC network for the experiments in this work on an Intel Core2 Duo processor-based system. Once trained, the resulting RC-based system can be used for real-time operation on moderate hardware since the computations are very fast (only matrix multiplications of small matrices).

For the supervised training of $\mathbf{W}_{\text{res}}^{\text{out}}$, we use fisher labeling[20] to get enhanced classification performance. Let $\hat{\mathbf{Y}}$ be a matrix containing the desired outputs where each line represents one output ($+1$ and -1) over time. As the number of positive desired outputs might be different from the number of negative desired outputs in each line, 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 $\hat{\mathbf{y}}^i$ sums up to 0:

$$\hat{y}^i_{\text{fish}}(t) = \begin{cases} \frac{n^i_+ + n^i_-}{n^i_+} & \text{if } \hat{y}^i(t) > 0 \\ -\frac{n^i_+ + n^i_-}{n^i_-} & \text{if } \hat{y}^i(t) < 0 \end{cases}, \tag{5}$$

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.

In the rest of this paper, we consider the following notations:

n_i	:	number of inputs
n_r	:	number of neurons in the reservoir
n_o	:	number of outputs
d_t	:	downsampling rate of the dataset
n_f	:	number of folds used in the cross-validation
$\alpha_1, \alpha_2, \alpha_3$:		leak rates for each neural pool in the reservoir

3 Robot Models

We use two robot models in the following experiments. Their respective simulation environments generate the data necessary for training the RC networks. The first model is part of the 2D SINAR simulator [2] and is described next. The environment of the robot is composed of several objects, each one of a

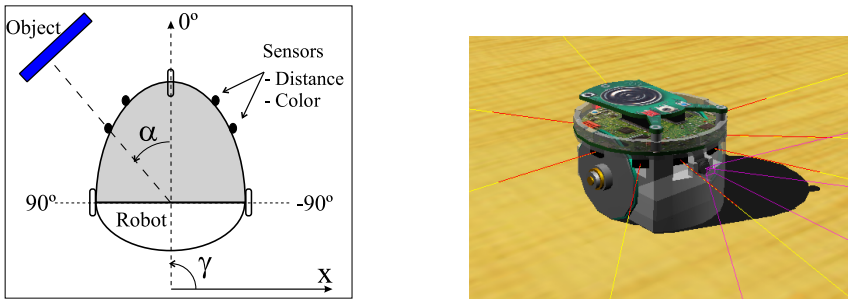


Fig. 2. Robot models used in the experiments. (a): robot model from SINAR simulator. (b): e-puck robot model.

particular color. Obstacles are represented by blue objects whereas targets are given by yellow objects. The robot model is shown in Fig. 2(a) . The robot interacts with the environment by distance and color sensors; and by one actuator which controls the movement direction (turning). Seventeen (17) sensor positions are distributed uniformly over the front of the robot (from -90° to +90°). Each position holds two virtual sensors (for distance and color perception) [2]. The distance sensors are limited in range (i.e., they saturate for distances greater than 300 distance units (d.u.)) and are noisy (they exhibit Gaussian noise on their readings, generated from $N(0, 60)$ in d.u.). A value of 0 means near some object and a value of 1 means far or nothing detected. 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 and the speed is constant (0.28 distance units (d.u.)/s).

The second model is the e-puck robot model [16]. We use the Webots simulation environment [15] for data generation which provides a physics model of the e-puck robot. The model is shown in Fig. 2(b). It has a 7 cm diameter. The e-puck is equipped with 8 infra-red sensors which measure ambient light and proximity of obstacles in a range of 4 cm originally. However, we change this range value in the simulator to 5 cm (for event detection experiments) and 15 cm (for robot localization experiments) in order to provide sufficient rich data for learning the respective tasks. This can be achieved by adding cheap infra-red range sensors to the real e-puck robot via an extension module. The actuators of the robot are 2 stepper motors and we limit its velocity to [0.6, 3] cm/s.

A comparison between both robot models is shown in Table 1. The robots from both models explore their environments according to specific controllers. The controller for the SINAR model (based on [2]) is a reactive system made of hierarchical neural networks which learn by interaction with the environment. Only already trained robot controllers, which all show very good exploratory behavior after training, are used for generating data. The controller for the e-puck model is made of a simple algorithm which follows predefined points from a trajectory in the environment. The robot speed for the event detection

task is set to either 0.63 cm/s or 1.25 cm/s depending on the proximity of obstacles (it can be 3 cm/s, 1.25 cm/s or 0.63 cm/s for the localization task).

The data from distance sensors and actuators collected from the robot simulator are used to train and test RC networks in a Matlab environment using the RCT Toolbox [3] [9].

4 Event Detection for Mobile Robots

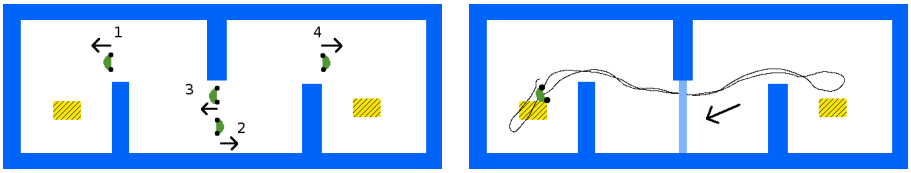
Event detection in noisy environments is not a trivial task. There can be very similar scenes from the robot's perspective so that precise event detection becomes very difficult to accomplish [8]. The goal here is to achieve efficient event detection using reservoir computing. The detection of events from raw sensory data is much related to the so called symbol grounding problem (or anchoring) in robotics [21]. Several applications are appealing in this context once deliberative robotic systems can benefit in several ways from efficient meaning extraction from sensory data [22–24].

Two experiments are conducted for the event detection task. The environments used for SINAR and e-puck are shown in Fig. 3(b) and Fig. 3(c), respectively. The first environment is 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). A blinking object located in the middle of the corridor (with random blink interval) can force the robot to change direction by blocking its way. In the second environment, the e-puck robot follows a predefined trajectory which continually visits the entire environment. Its trajectory can be changed when it reaches the middle of the environment (dotted line in Fig. 3(c)) with a

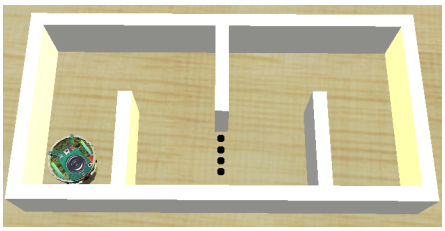
[3] This is an open-source Matlab toolbox for Reservoir Computing which is freely available at http://www.elis.ugent.be/rct

Table 1
Robot models

	SINAR model	e-puck model
No. Dist. Sensors	17	8
Range of Dist. Sens.	300 d.u.	5 cm or 15 cm
Noise on sensors	N(0,60 d.u.)	30%
Speed	0.28 d.u.	3 cm/s, 1.25 cm/s or 0.63 cm/s
Physics model	no	yes



(a) Definition of the 4 events (b) SINAR environment E1



(c) e-puck environment P1

Fig. 3. Environments used for the event detection task. (a) Four events are labeled and shown graphically (by arrows). (b) SINAR environment with a *blinking* obstacle in the middle of the corridor, indicated by an arrow. A typical robot trajectory (after controller learning) can be seen in the figure. Two boxes in the environment are used as targets for the robot. (c) e-puck simulation environment (the 4 events are defined similarly). The dotted line represents a decision point which makes the robot cross the line or go back with equal probabilities.

probability of 50%.

There are four possible events of predefined duration and location, which are labeled in Fig. 3(a). The interpretation should be: when the robot passes through a predefined location with a specific heading, an event should be detected (e.g. entering the corridor corner area, passing through the middle of the corridor).

Experiment 1 is accomplished considering the SINAR environment E1 and experiment 2 takes place in the e-puck environment P1. Experiments 1 and 2 have 126.000 and 120.000 timesteps of simulation time, respectively. Parameter configuration for both experiments are shown in Table 2).

In order to match the dynamics of the sensory input to the temporal dynamics of the reservoir, we make use of both data downsampling (d_t) and leak rate (α) in the reservoir. Although resampling and leak rate are considered equivalent [17], it seems that the combination of both methods yields superior

Table 2
Parameter configuration for event detection

Model	n_i	n_o	d_t	α	n_r	n_f	$\mathbf{W}_{\text{inp}}^{\text{res}}$	Training
SINAR	18	4	20	0.6	800	7	$\{\pm 0.2, 0\}$	Ridge regression
e-puck	10	4	15	0.8	800	8	$\{\pm 0.2, 0\}$	Ridge regression

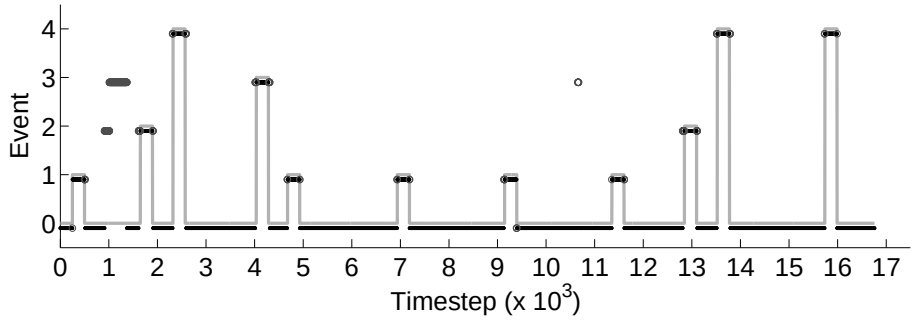
performance (as it will be shown in the results) by more finely adjusting the temporal dynamics of the reservoir to the input signal's dynamics. The value of the parameters d_t and α was optimized by performing a grid search (the combination of parameters with highest test performance was chosen).

The inputs to the network for the SINAR model are 17 distance sensors and 1 robot actuator (direction adjustment) while for the e-puck model the inputs are composed of 8 sensors and 2 motor actuators. The reservoir size is 800 neurons for all experiments in this work, although smaller reservoirs (e.g., 200 or 400 nodes) can already perform very good. 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 readouts output a negative value, then the winner-take-all function set every output to -1 (this means no event is detected). The connection matrix from input to the reservoir ($\mathbf{W}_{\text{inp}}^{\text{res}}$) is initialized to -0.2, 0.2 and 0 with probabilities 0.15, 0.15 and 0.7, respectively. This parameter setting for weight matrices are not critical for the tasks in this work.

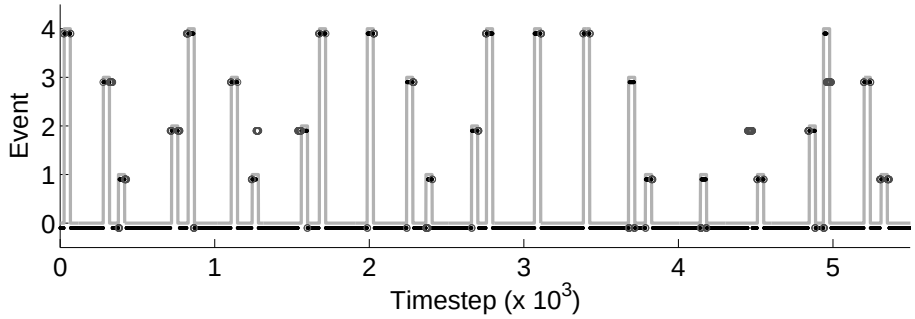
The performance measure considers the number of correctly predicted observations and uses a 7-fold (8-fold) cross-validation method for the SINAR model (e-puck). It is important to note that if the dataset is resampled, then the output of the network is upsampled to the original sampling rate of the dataset so that the performance is correctly calculated (differently from [14]). Additionally we also calculate the percentage of correctly detected events for each of the 4 possible events separately.

The results are shown in Fig. 4. For both robot models, a RC network performs very good by achieving 95.4% and 93.1% of classification performance on test data. Although the events in the run are not periodic, the 4 classes of events are correctly detected during all the simulation, with few mispredictions. Most of the errors are explained by the temporal resolution of the RC-based detector, that is, the reservoir is sometimes not accurate enough in the very beginning/end of an event (i.e., in the temporal boundary of events). This problem has to do with the downsampling of the input signal (for matching the reservoir dynamics) which is upsampled for performance measure. So, part of the temporal resolution is lost in this process of downsampling/upsampling. First investigations on reservoirs using only leak rate (without resampling the input) yield equivalent temporal resolution, showing that it is difficult to get perfect temporal resolution.

In Fig. 5, we can see how the downsampling rate of the dataset (d_t) and reservoir size influence the test performance on the event detection task using the e-puck robot model. If a dataset is downsampled by $d_t = 10$, for instance, the resulting dataset will be $1/10$ smaller than the original one, effectively slowing



(a) Event detection using SINAR model



(b) Event detection using e-puck model

Fig. 4. Event detection performed by RC network. The gray solid line represents the actual event whereas the black points are the predicted events. Mispredictions are marked with circles. (a) Using SINAR simulation model (performance of 95.4 % on this test data) (b) Event detection using e-puck simulation model (performance of 93.1 % on this test data).

Table 3
Results for event detection

Model	Timesteps	Train Perf.	Test Perf.	Perf. Events 1, 2, 3 and 4 resp.			
SINAR	126 K	94.7 %	93.2 %	95.5%	95.6%	100%	99.6%
e-puck	120 K	93.2 %	92.3 %	86.9%	97.2%	96.7%	82.6%

down the input signal. Fig. 5 shows that a downsampling of 15 timesteps is the optimal choice. It is possible to observe that when d_t is bigger than 30 timesteps, as the downsampling rate increases the performance deteriorates. The figure also shows that bigger reservoirs (with more neurons) have more memory (while require less resampling), thus increasing performance.

Statistics on experiments 1 and 2 are given in Table 3. Each experiment is evaluated 30 times with different stochastically generated reservoirs and the results are averaged over these 30 runs. The table shows that the results are consistent, with 93.2% and 92.3% of performance on test data for SINAR and e-puck models, respectively.

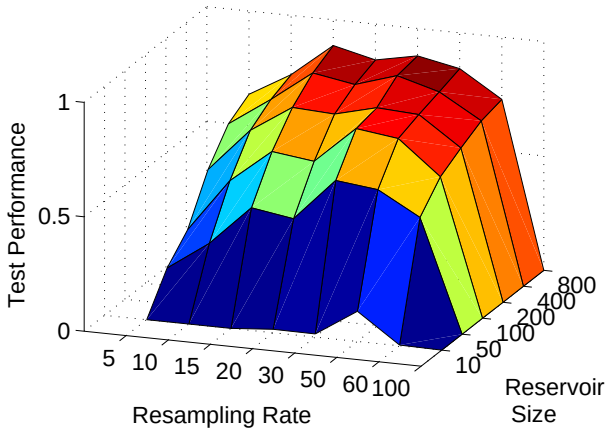


Fig. 5. Resampling rate and reservoir size vs. test performance. Each point of the plot is the mean performance (correct classifications) over 30 runs for the event detection task using the e-puck robot model.

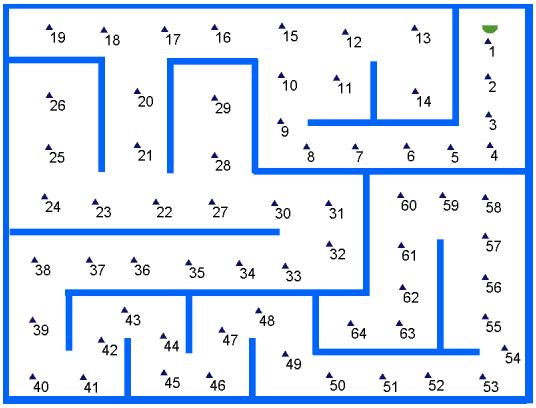
5 Robot Localization

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

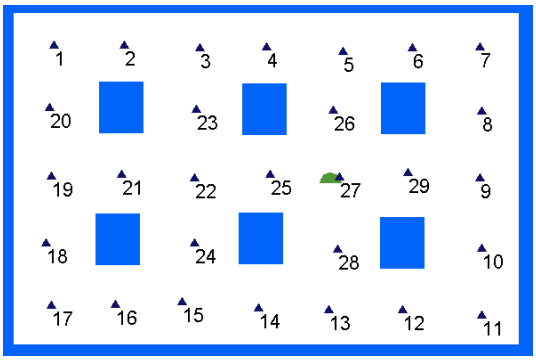
Localization (or position detection) for mobile robots is usually computationally expensive in terms of space and time requirements [4]. Traditional algorithms are based on explicit maps which must be constructed before robot localization is possible. This section shows how a reservoir can be used for robot localization. Similar work which uses a Long-Short Term Memory RNN for this task is described in [25]. In this section, we show that localization is achieved considering two abstract concepts: locations and rooms (in [25], only rooms are considered).

The first investigations were made with the SINAR model. Two maze-like environments are used for the robot localization task (see Fig. 6). The first environment contains 64 predefined locations, that are displayed by small triangles labeled by numbers. The second environment has 29 locations distributed in a symmetric map.

The parameter configuration used for the following experiments is shown in Table 4. The size of the readout output layer (n_o) is equivalent to the number of predefined locations in the environment. 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



(a) SINAR Environment E2



(b) SINAR Environment E3

Fig. 6. Environments used for the experiments. The first environment is tagged with 64 labels displayed by small triangles. The second environment has 29 labels distributed through very similar areas.

previous section). The connection matrix from input to the reservoir ($\mathbf{W}_{\text{inp}}^{\text{res}}$) is initialized to -0.1, 0.1 and 0 for SINAR model (-0.9, 0.9 and 0 for e-puck) with probabilities 0.15, 0.15 and 0.7, respectively.

Experiment 3 is accomplished with the first environment from Fig. 6 and lasts 180.000 timesteps. The resulting robot occupancy grid can be seen in Fig. 7(a): it shows that the reservoir is predicting the robot location very well (the performance is 91.6% on test data), with very few mispredictions. Experiment 4 uses the same environment E2 with 11 additional slow moving obstacles distributed throughout the environment. These dynamic objects change the robot's behavior and also add more noise to sensor readings. The simulation has 180.000 timesteps. The respective occupancy grid in Fig. 7(b) shows that the reservoir is correct in most of the predictions (81.1 %). Some of the mispredictions are located a bit further from the actual position, due to the new source of dynamics and noise, although they generally tend to be very short.

Experiment 5, accomplished in environment E3 (Fig. 6(b)), represents a new

Table 4
Parameter configuration for localization

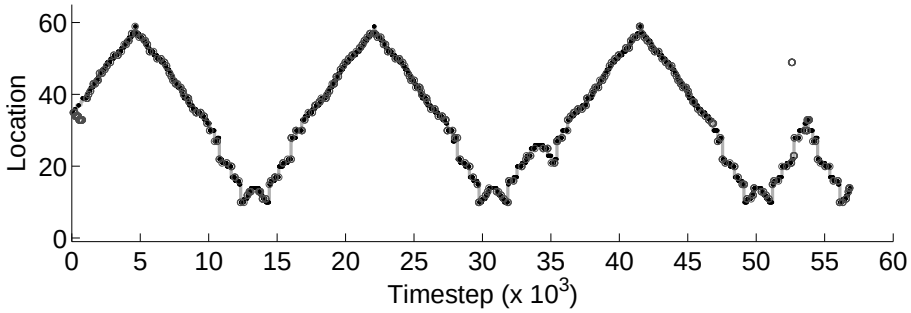
Model	n_i	n_o	d_t	α_1	α_2	α_3	n_r	n_f	$\mathbf{W}_{\text{inp}}^{\text{res}}$	Training
SINAR	18	{64,29}	50	0.6	–	–	800	3	$\{\pm 0.1, 0\}$	Least Sq.
e-puck$_{loc}$	10	30	10	0.05	0.8	1	800	7	$\{\pm 0.9, 0\}$	Rid.regr.
e-puck$_{room}$	10	4	20	0.05	0.8	1	800	5	$\{\pm 0.9, 0\}$	Rid.regr.

challenge for the reservoir-based position detector: the environment has several symmetries and identical areas. For instance, going from position 27 to 26 looks identical to the robot as going from position 22 to 24. The simulation has 150.000 timesteps. The resulting occupancy grid in Fig. 7 shows an efficient position detector, featuring a performance of 89.1 % of correct predictions on test data.

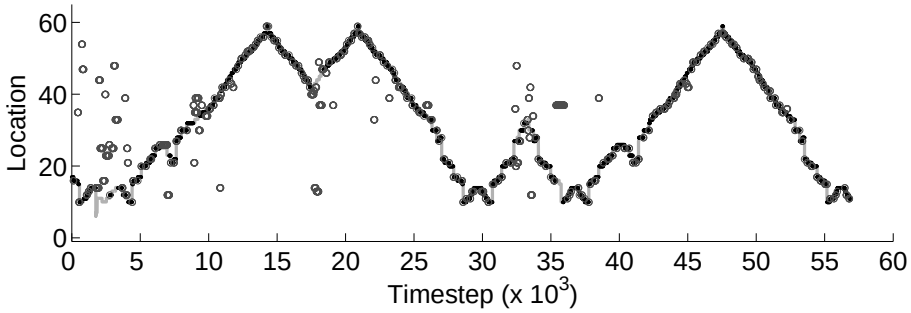
The following experiments are done with the e-puck robot model. The environment used is shown in Fig 8(a). It is composed of 4 big rooms with doors connecting them. Fig 8(b) shows 30 points distributed in the map which are connected by lines representing possible robot paths between them. When the robot reaches a point which can lead to 2 other possible points, the robot controller decides to choose one of the points with equal probabilities. In this way, the robot may stay in room 3 for varying periods of time in the same run, for instance. The localization task is to detect which one of the 30 points the robot is most close to. The room detection task uses the map in Fig 8(c) which shows the boundaries dividing the rooms.

The parameter configuration for both location and room detection is shown in Table 4. The experiments with the e-puck consider 3 neural pools in the reservoir, each with distinct leak rates $(\alpha_1, \alpha_2, \alpha_3)$. This feature makes the reservoir work in several timescales, thus, making it more efficient when the task considers a robot with a varying speed (in our case, the robot can have 3 different velocities). The results in Fig. 9 show that the test performance reaches 84% of correct classification when 3 distinct leak rates are used, an increase of at least 6% compared to the experiment with no leak considered (several mispredictions present in Fig. 9(a) are removed by the new setup in Fig. 9(b)).

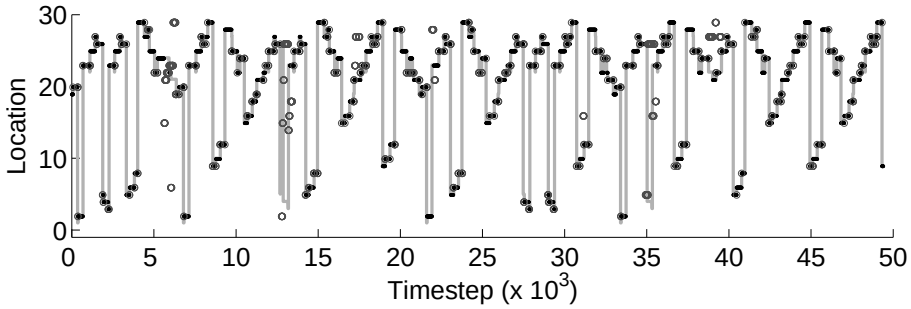
A summary of the localization experiments with associated results are shown in Table 5. This table presents additional results from experiments considering only dataset resampling (without the use of leak rates) so that we can reliably draw conclusions. Each experiment is evaluated 30 times with different stochastically generated reservoirs and the results are averaged over these 30 runs. We can observe that the use of leak rates yields the greatest increases in performance for the experiments with the SINAR model in environment E2^{mov}(with moving obstacles) and with the e-puck robot (in this case, 3 neu-



(a) Location Detection (E2)



(b) Location Detection (E2 with 11 moving obstacles)

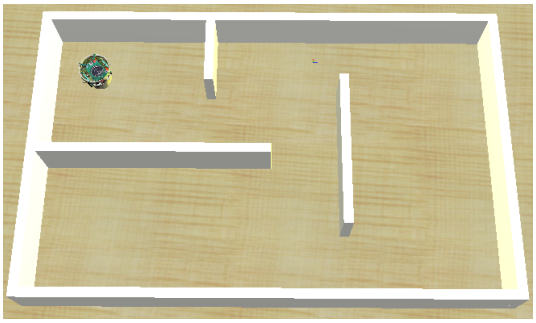


(c) Location Detection (E3)

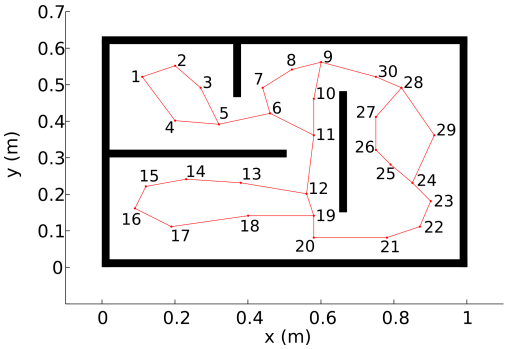
Fig. 7. Robot occupancy grids showing the predicted location (on test data, that is 1/3 of the total data) and the actual robot positions (solid gray line). Mispredicted locations are represented by a circle. (a) Experiment in environment E2 (test performance of 91.6% of correct detection). (b) Experiment in environment E2 with 11 slow moving obstacles (test performance of 81.1% of correct detection). (c) Experiment in environment E3 (test performance of 89.1% of correct detection).

ral pools of distinct leak rates).

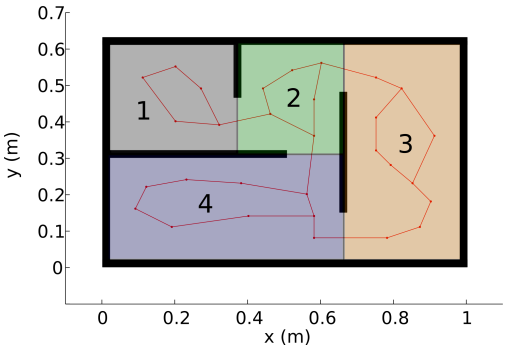
Room detection can also be achieved in a similar way. The results are shown in Fig. 10 using the configuration from Table 4. The RC-based room detector is very efficient during more than 7000 timesteps: it shows a performance of 93.6% on test data. There are few mispredictions, and most of them are located in the temporal boundaries between one room and the next one, which is a result from the downsampling/upsampling artifact. Table 6 shows the statistics for the room detection task. Each one of the rooms are correctly



(a) Environment for e-puck (P2)



(b) Map for Location detection



(c) Map for Room detection

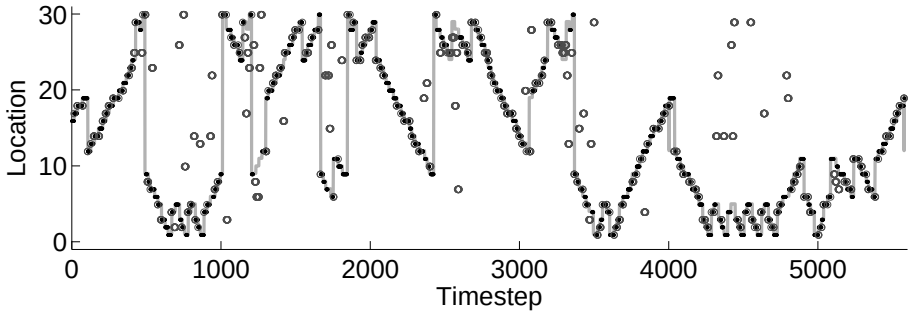
Fig. 8. Environment used for localization experiments with the e-puck robot model. (a): the environment used. (b) the map of the environment with the locations to be detected. (c) the map of the environment showing the 4 rooms to be detected and the robot exploring trajectory.

Table 5
Results for location detection

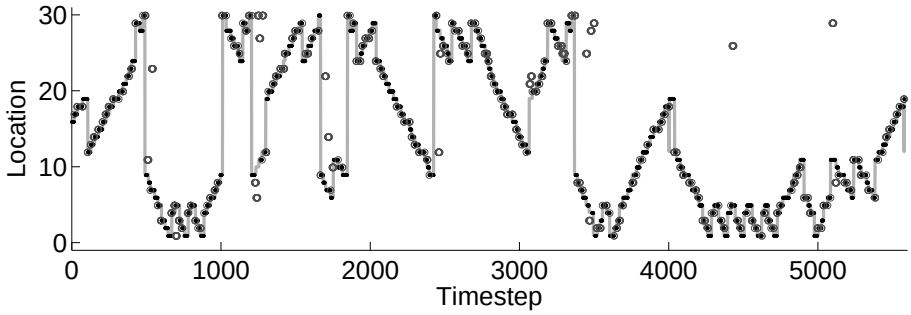
Model (Env)	Leak	Timesteps	Train Perf.	Test Perf.	Std Test
SINAR (E2)	no	180 K	94.0 %	**89.2 %**	0.4%
SINAR (E2)	yes(1)	180 K	94.3 %	**90.7 %**	0.1%
SINAR (E3)	no	150 K	94.4 %	**88.6 %**	0.3%
SINAR (E3)	yes(1)	150 K	94.4 %	**89.1 %**	0.3%
SINAR ($E2^{mov}$)	no	180 K	93.0 %	**76.0 %**	0.7%
SINAR ($E2^{mov}$)	yes(1)	180 K	94.2 %	**79.1 %**	0.5%
e-puck	no	40 K	87.2 %	**78.9 %**	0.4%
e-puck	yes(3)	40 K	90.4 %	**85.1 %**	0.5%

detected by a rate of at least 91%.

Experiments only considering distance sensors (removing actuator) result in similar performance reported for the previous experiments in this section. The reservoir network also copes with the kidnapping situation (also reported in



(a) No leak rate considered



(b) 3 neural pools of distinct leak rates

Fig. 9. Location detection performed by a RC network (e-puck). (a) A normal reservoir gives a performance of 77.7 % on test data (b): A reservoir containing 3 neural pools of distinct leak rates yields a performance of 84 % (test data). The gray solid line represents the actual location whereas the black points are the predicted location. Mispredictions are marked with circles.

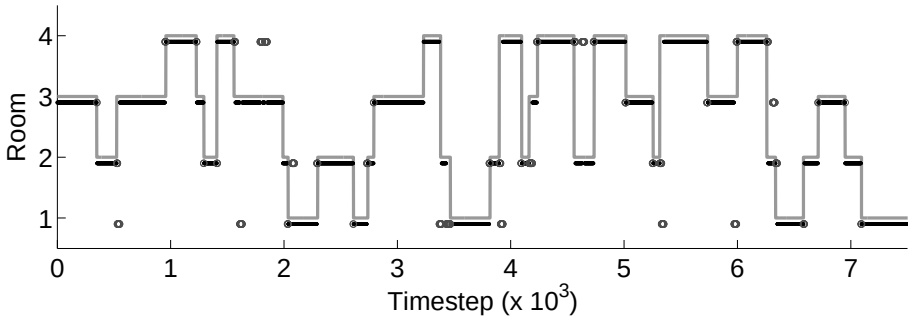


Fig. 10. Room detection performed by a RC network (e-puck). The points in the plot (rooms) are correctly detected with a rate of 93.6 % (test data). The gray solid line represents the actual room whereas the black points are the predicted room. Mispredictions are marked with circles.

Table 6
Results for room detection

Model	Timesteps	Train Perf.	Test Perf.	Perf. Rooms 1, 2, 3 and 4 resp.			
e-puck	39 K	97.2 %	93.1 %	95.4%	91.5%	91.8%	93.1%

[25]). In a new experiment using environment E2, the robot is replaced from

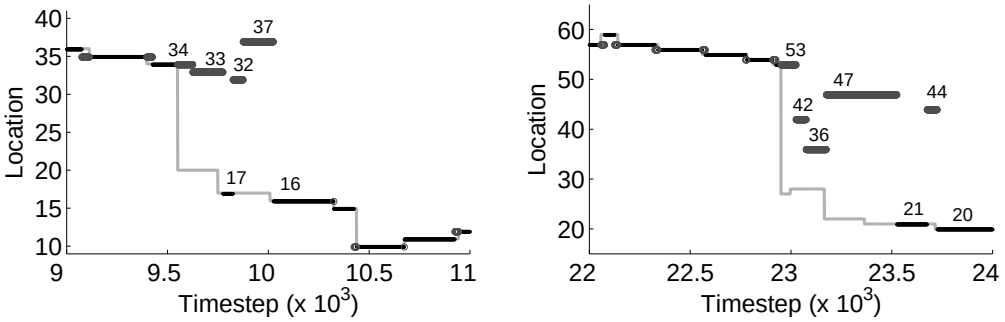


Fig. 11. Occupancy grid after kidnapping the robot in environment E2 of Fig. 6. The solid gray line represents the actual robot position. Correct predictions are given by black points while wrong predictions are marked with circles. The predictions of the RC network are labeled with numbers after the robot is kidnapped. (a) At time step 9551, the robot is moved from position 34 to position 20. The reservoir network predicts successfully the current robot position when the robot is in location 16. The robot visits 2 locations (20 and 17) until the successful prediction. (b) At time step 22951, the robot is moved from position 53 to position 27. The reservoir network predicts successfully the current robot position when the robot is in location 21. The robot visits 3 locations (27, 28 and 22) until the successfull prediction.

location 34 to location 20 (see Fig. 11(a)). The network is able to successfully predict the robot position when the robot reaches location 16. Note that the predicted locations following the kidnapping are near location 34, as if the robot kept the original path. After some timesteps, the reservoir realizes it is actually in location 16, given the history of sensory inputs since the kidnapping. Another example is given in Fig. 11(a), where the robot is kidnapped from location 53 to location 27. Note that the RC network is not trained for the kidnapping situation.

6 Discussions

In this work we show that it is possible to detect complex events and perform robot localization in complex and even dynamic environments using a fixed, randomly created dynamic system which is processed by a trained linear readout layer. Only a limited number of noisy and low range sensors are needed for building successful RC-based detectors. The proposed system shows very good performance in difficult environments such as mazes or environments which are highly symmetric. The detection of events and locations is achieved through a linear classification of the temporal dynamics existing in the fixed dynamic system which in turn is completely determined by the anterior stream of the sensory inputs. Thus, it is the short-term memory capabilities of the random reservoir and its non-linear projections which allow the system to perform efficient event detection and localization. The actual behaviors driving the robot

were not used as inputs to the system (as in [12]), but only low-level motor commands (only distance sensory inputs are sufficient for the tasks however).

When a RC-based system learns to perform robot localization, an implicit map is formed by the reservoir's dynamics in combination with the trained readout. This might seem hard to grasp, but in [26] we showed that we can run a trained RC-based localization system *backwards*, and create an explicit representation of the map that is implicitly stored in the reservoir. The inputs of the reverse system are locations, and the output are predicted sensor range measurements. In this way, we can reconstruct the map of the environment by driving the robot through different locations and recording what the reservoir *has in mind* at the respective moment.

We show that event detection and localization works on a physically realistic simulated robot model. As future work, we plan to implement both techniques on the real robotic platform, as it is considered the standard and best evaluation method for robotic systems. Furthermore, 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 (e.g. information about events, locations and rooms).

The importance of timescales in reservoir systems was clearly demonstrated in this work, especially when the robot's speed is not constant. From an RC view, we could further improve the performance, for example, by using a so called bandpass reservoir [27], for example. This idea consists of adding bandpass filters to the reservoir, making it possible to capture a wide range of timescales inside a single reservoir. This setup could greatly improve the reservoirs performance on tasks with a wide range of speeds.

Future work also includes the unsupervised detection and generation of locations, much resembling the creation of place cells in the hippocampus. The current setup is trained in a completely supervised way. This is not realistic from the perspective of intelligent autonomous systems. The current technique can however be combined with approximate dead reckoning and on-line learning of novel locations to form a truly unsupervised localization and mapping system.

Acknowledgments

This research is partially funded by FWO Flanders project G.0317.05.
We would like to thank Karel Braeckman for helping to generate the e-puck datasets from the Webots simulation environment.

References

[1] R. Arkin, Behavior-based robotics, MIT Press, 1998.

[2] E. A. Antonelo, A.-J. Baerlvedt, T. Rognvaldsson, M. Figueiredo, Modular neural network and classical reinforcement learning for autonomous robot navigation: Inhibiting undesirable behaviors, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Vancouver, 2006, pp. 498– 505.

[3] J. Guivant, E. Nebot, S. Baiker, Autonomous navigation and map building using laser range sensors in outdoor applications, Journal of Robotics Systems 17 (10) (2000) 565–583.

[4] T. Bailey, H. Durrant-Whyte, Simultaneous localisation and mapping (SLAM): Part ii state of the art, Robotics and Automation Magazine.

[5] H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks, Tech. Rep. GMD Report 148, German National Research Center for Information Technology (2001).

[6] W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, Neural Computation 14 (11) (2002) 2531–2560.

[7] J. J. Steil, Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Vol. 1, 2004, pp. 843–848.

[8] H. Jaeger, Short term memory in echo state networks, Tech. Rep. GMD Report 152, German National Research Center for Information Technology (2001).

[9] D. Verstraeten, B. Schrauwen, M. D'Haene, D. Stroobandt, A unifying comparison of reservoir computing methods, Neural Networks 20 (2007) 391–403.

[10] H. Jaeger, H. Haas, Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication, Science 308 (2004) 78–80.

[11] K. Schönherr, M. Cistelecan, J. Hertzberg, T. Christaller, 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, p. 305319.

[12] J. Hertzberg, H. Jaeger, F. Schönherr, Learning to ground fact symbols in behavior-based robots, in: Proceedings of the 15th European Conference on Artificial Intelligence, 2002, pp. 708–712.

[13] J. O'Keefe, J. Dostrovsky, The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat, Brain Research 34 (1971) 171–175.

[14] E. A. Antonelo, B. Schrauwen, X. Dutoit, D. Stroobandt, M. Nuttin, Event detection and localization in mobile robot navigation using reservoir computing, in: J. M. de Sa et al. (Ed.), ICANN, Part II, Springer-Verlag, 2007, pp. 660–669.

[15] O. Michel, Webots: Professional mobile robot simulation, Journal of Advanced Robotics Systems 1 (1) (2004) 39–42.

[16] e-puck, http://www.e-puck.org/, e-puck education robot (2007).

[17] B. Schrauwen, J. Defour, D. Verstraeten, J. Van Campenhout, The introduction of time-scales in reservoir computing, applied to isolated digits recognition, in: Proceedings of the International Conference on Artificial Neural Networks (ICANN), 2007.

[18] H. Jaeger, M. Lukosevicius, D. Popovici, Optimization and applications of echo state networks with leaky integrator neurons, Neural Networks 20 (2007) 335–352.

[19] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer, 2006.

[20] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification - Second Edition, John Wiley and Sons, Inc., 2001.

[21] S. Harnad, The symbol grounding problem, in: Encyclopedia of Cognitive Science, London: Nature Publishing Group/Macmillan, 2003.

[22] S. Harnad, Grounding symbols in the analog world with neural nets a hybrid model, Psychology 12 (2001) 12–78.

[23] P. Vogt, Symbol grounding in communicative mobile robots, in: S. Coradeschi, A. Saffiotti (Eds.), Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems: Papers from the 2001 AAAI Fall Symposium, AAAI Press, 2001, pp. 87–94.

[24] M. Rosenstein, P. R. Cohen, Concepts from time series, in: Proceedings of the Fifteenth National Conference on Artificial Intelligence, 1999, pp. 739–745.

[25] A. Forster, A. Graves, J. Schmidhuber, RNN-based learning of compact maps for efficient robot localization, in: Proceedings of ESANN, 2007.

[26] E. A. Antonelo, B. Schrauwen, J. Campenhout, Generative modeling of autonomous robots and their environments using reservoir computing, Neural Process. Lett. 26 (3) (2007) 233–249.

[27] F. Wyffels, B. Schrauwen, D. Verstraeten, D. Stroobandt, Band-pass reservoir computing, submitted to IJCNN 2008 (2008).