Anais do IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional (IX CBRN)
Ouro Preto 25-28 de Outubro de 2009

ON DIFFERENT LEARNING APPROACHES WITH ECHO STATE NETWORKS FOR LOCALIZATION OF SMALL MOBILE ROBOTS

Eric Aislan Antonelo[1]*, Benjamin Schrauwen*

*Electronics and Information Systems Department
Ghent University, Ghent, Belgium

Emails: eric.antonelo@elis.ugent.be, benjamin.schrauwen@elis.ugent.be

Abstract— Animals such as rats have innate and robust localization capabilities which allow them to navigate to goals in a maze. The rodent's hippocampus, with the so called place cells, is responsible for such spatial processing. This work seeks to model these place cells using either supervised or unsupervised learning techniques. More specifically, we use a randomly generated recurrent neural network (the reservoir) as a non-linear temporal kernel to expand the input to a rich dynamic space. The reservoir states are linearly combined (using linear regression) or, in the unsupervised case, are used for extracting slowly-varying features from the input to form place cells (the architectures are organized in hierarchical layers). Experiments show that a small mobile robot with cheap and low-range distance sensors can learn to self-localize in its environment with the proposed systems.

Keywords— robot localization, place cells, hippocampus, mobile robots, reservoir computing.

1 Introduction

The most standard approach to robot localization is situated within the group of probabilistic methods such as SLAM (Simultaneous Localization And Mapping) which is used for mobile robots usually equipped with expensive laser scanners. The SLAM method can under suitable assumptions (Thrun et al., 2005) efficiently build a map of the environment while the robot navigates in it.

From biology, it is known that the hippocampus region of the brain is responsible for storing events as well as spatial information from the environment (Moser et al., 2008). Studies carried on rats show that some cells in the hippocampus, called place cells, learn to respond in accordance with the rat location in its environment. A place cell has a peak activity when the rat is located at a specific area of the environment, i.e., at the place field of the cell. On the other hand, grid cells code for multiple locations which are spatially distributed in a grid. It is assumed that place cells can be formed based on the processing of grid cells (Moser et al., 2008).

The current work investigates the application of biologically inspired architectures to the problem of learning the localization ability for a mobile robot in an unstructured environment. Robustness, learning and low computation time are some characteristics of these biological inspired systems. Most systems are based on visual input from camera (Arleo et al., 2004; Franzius et al., 2007; Stroesslin et al., 2005; Milford et al., 2007) and models hippocampal place cells from rats (Arleo et al., 2004; Franzius et al., 2007; Stroesslin et al., 2005; Milford et al., 2007).

In this paper, we use Echo State Networks (ESN) (Jaeger, 2001) as a tool for learning the desired task. These networks, known also as Reservoir Computing (RC) networks (Schrauwen et al., 2007), are composed of a randomly generated and non-trainable Recurrent Neural Network (RNN), the reservoir, and a readout output layer which is trained by standard linear regression methods, in a supervised approach. The training is simple and does not suffer from convergence problems, as the reservoir itself is not trained. Fig. 1 shows the architecture of an ESN. In this work, the ESN is used in conjunction with two different learning paradigms: supervised and unsupervised. Usually RC systems are trained in a supervised way. Here we also present an architecture which learns the slow features from the input data in an unsupervised way as in (Antonelo and Schrauwen, 2009). We use the reservoir, in the first lower layer, as a non-linear temporal kernel which expands the input space to a high dimensional space. The upper layer in the architecture learns using the Slow Feature Analysis (SFA) algorithm (Wiskott and Sejnowski, 2002) which models roughly the grid cells in the hippocampus. The last upper layer learns by Independent Component Analysis (ICA) (Hyvärinen and Oja, 2000), implementing a sparse coding on the SFA output to form the hippocampal place cells.

Our robot model has only 8 limited-range distance

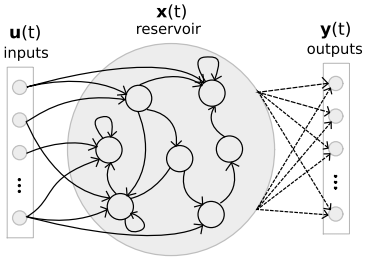


Figure 1: Echo State Network. Solid and dashed lines represent fixed and trainable connections, respectively.

[1]Eric Antonelo is supported by the Special Research Fund from Universiteit Gent (BOF).

sensors, which are the only input used for the localization task. Using this scheme, we show that it is possible to learn, either in a supervised or unsupervised way, place cells which can efficiently localize a real robot. This work is organized as follows. In Section 2, we present the ESN approach as well as the unsupervised learning algorithms SFA and ICA used in our proposed architectures. Experiments and results are shown in Section 3 and the conclusion in Section 4.

2 Methods

2.1 Echo State Network

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 (Fig. 1). The state update equation for the reservoir and the readout output equation are as follows:

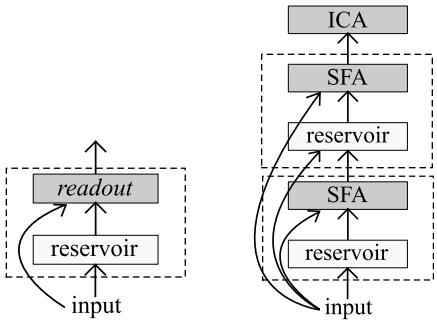
$$\mathbf{x}(t+1) = f(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}}). \quad (1)$$

$$\mathbf{y}(t+1) = \mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{x}(t+1) + \mathbf{W}^{\mathrm{o}}_{\mathrm{b}}. \quad (2)$$

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 (commonly used for ESNs). The matrices $\mathbf{W}$ represent the connection weights between the nodes of the complete network, where r, i, o, b denotes *reservoir*, *input*, *output*, and *bias*, respectively. All weight matrices representing the connections to the reservoir (denoted as $\mathbf{W}^{\mathrm{r}}_{\cdot}$) are initialized randomly, while all connections to the output layer (denoted as $\mathbf{W}^{\mathrm{o}}_{\cdot}$) are trained. The initial state is $\mathbf{x}(0) = \mathbf{0}$. The random generation of $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$ is such that the largest absolute eigenvalue of $\mathbf{W}$, the spectral radius $|\lambda_{max}|$, is less than 1. In this way, the randomly generated reservoir weights ($N(0,1)$) are rescaled such that the system is not chaotic, but its dynamic regime is situated at the edge of stability (Jaeger, 2001). The value for $|\lambda_{max}|$ and other matrices $\mathbf{W}^{\mathrm{r}}_{\mathrm{i}}$ and $\mathbf{W}_{\mathrm{b}}$ are given in Section 3.

2.2 Supervised Learning

Supervised learning is the usual training approach for ESNs. Architecture 1 in Fig. 2(a) illustrates the same network in Fig. 1. Consider that the total number of time samples of the training dataset is n_s. Training is performed using linear regression on the reservoir states. For this, the reservoir is driven by an input sequence $\mathbf{u}(1), \ldots, \mathbf{u}(n_s)$ (robot sensors) which yields a sequence of states $\mathbf{x}(1), \ldots, \mathbf{x}(n_s)$ using (1). In this process, state noise can be added to (1) for regularization purposes, The generated states are collected row-wise into a matrix $\mathbf{M}$ of size $n_s \times (n_r + 1)$ where the last column of $\mathbf{M}$ is composed of 1's (representing the bias). The desired teacher outputs (robot location)



(a) Architecture 1 (b) Architecture 2

Figure 2: Reservoir architectures. Shaded layers (readout, SFA and ICA layers) can be trained whereas white layers (input and reservoir) have fixed non-trainable weights.

are collected row-wise into a matrix $\hat{\mathbf{Y}}$ (there is an output unit for each location). Then, the readout output's matrix $\mathbf{W}^{\mathrm{o}}_{\mathrm{rb}}$ (i.e., the column-wise concatenation of $\mathbf{W}^{\mathrm{o}}_{\mathrm{r}}$ and $\mathbf{W}^{\mathrm{o}}_{\mathrm{b}}$) of size $(n_r + 1) \times n_o$ is created by solving (in the mean square sense):

$$\mathbf{M}\mathbf{W}^{\mathrm{o}}_{\mathrm{rb}} = \hat{\mathbf{Y}} \quad (3)$$

$$\mathbf{W}^{\mathrm{o}}_{\mathrm{rb}} = (\mathbf{M}^{\top}\mathbf{M})^{-1}\mathbf{M}^{\top}\hat{\mathbf{Y}} \quad (4)$$

2.3 Unsupervised Learning

Slow Feature Analysis

In order to autonomously learn the localization ability for a mobile robot, we need to use an unsupervised learning mechanism which can extract significant information from only 8 distance sensors. For this, we use Slow Feature Analysis (SFA) to learn the slowly-varying features of the reservoir states which in turn is stimulated by the distance sensors. SFA, introduced in (Wiskott and Sejnowski, 2002), is based on the concept of temporal slowness for learning invariant or slowing varying signals from the input data. In (Franzius et al., 2007), an hierarchy of SFA layers is used for modeling grid cells from the entorhinal cortex of rats and also hippocampal place cells.

Given a high-dimensional input signal $\mathbf{v}(t)$, find a set of scalar functions $g_i(\mathbf{v}(t))$ so that the SFA output $y_i = g_i(\mathbf{v}(t))$ varies as slowly as possible and still carries significant information. Mathematically, find SFA output signals $y_i = g_i(\mathbf{v}(t))$ such that (Wiskott and Sejnowski, 2002):

$$\Delta(y_i) := \langle \dot{y}_i^2 \rangle_t \quad \text{is minimal} \quad (5)$$

under the constraints

$$\langle y_i \rangle_t = 0 \quad \text{(zero mean)} \quad (6)$$

$$\langle y_i^2 \rangle_t = 1 \quad \text{(unit variance)} \quad (7)$$

$$\forall j < i, \langle y_i y_j \rangle_t = 0 \quad \text{(decorrelation and order)} \quad (8)$$

where $\langle . \rangle_t$ and $\dot{y}$ represent temporal averaging and the derivative of y, respectively.

Learning Algorithm: Before applying the learning algorithm, the input signal $\mathbf{v}(t)$ is normalized for having zero mean and unit variance. We do not consider the common non-linear expansion (Wiskott and Sejnowski, 2002) used as a preprocessing step, once the reservoir already temporally expands the input signal. So, $g_i(\mathbf{v}) = \mathbf{w}^T\mathbf{v}$. The SFA algorithm is defined as:
Solve the generalized eigenvalue problem:

$$\mathbf{A}\mathbf{W}_{\text{sfa}} = \mathbf{B}\mathbf{W}_{\text{sfa}}\Lambda, \tag{9}$$

where $\mathbf{A} := \langle \dot{\mathbf{v}}\dot{\mathbf{v}}^T \rangle_t$ and $\mathbf{B} := \langle \mathbf{v}\mathbf{v}^T \rangle_t$.
The eigenvectors $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{n_{sfa}}$ corresponding to the ordered generalized eigenvalues $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{n_{sfa}}$ solve the learning task defined above, satisfying (6-8) and minimizing (5) (see (Wiskott and Sejnowski, 2002) for more details). This procedure is guaranteed to find the global optimum.

Independent Component Analysis

The learning task for Independent Component Analysis (ICA) can be defined as follows. Suppose that a linear mixture of signals $q_1, q_2 ... q_n$ can be used for finding the n independent components or latent variables $s_1, s_2 ... s_n$. The observed values $\mathbf{q}(t) = [q_1(t), q_2(t) ... q_n(t)]$ can be written as:

$$\mathbf{q}(t) = \mathbf{A}\mathbf{s}(t) \tag{10}$$

where $\mathbf{A}$ is the mixing matrix; and $\mathbf{s}(t) = [s_1(t), s_2(t) ... s_n(t)]$ is the vector with the independent components (both $\mathbf{A}$ and $\mathbf{s}(t)$ are assumed to be unknown). After finding an estimate for matrix $\mathbf{A}$, vector $\mathbf{s}(t)$ can be written as:

$$\mathbf{s}(t) = \mathbf{W}_{\text{ica}}\mathbf{q}(t) \tag{11}$$

where $\mathbf{W}_{\text{ica}}$ is the inverse matrix of $\mathbf{A}$. The fundamental assumption for ICA is that the components s_i are statistically independent and have nongaussian distributions (Hyvärinen and Oja, 2000).

Learning algorithm: In this work the matrix $\mathbf{W}_{\text{ica}}$ is found with the FastICA algorithm (Hyvärinen and Oja, 2000). Before using ICA, the observed vector $\mathbf{q}(t)$ is preprocessed by centering (zero-mean) and whitening (decorrelation and unit variance). FastICA uses a fixed-point iteration scheme for finding the maximum of the nongaussianity of $\mathbf{w}\mathbf{q}(t)$ (where $\mathbf{w}$ is a weight vector of one neuron). This means that the independent components will mostly be clustered, concentrated on specific values, in contrast with the more random values of Gaussian variables. The next units $\mathbf{w}_i$ in $\mathbf{W}_{\text{ica}}$ are found one by one such that the outputs $\mathbf{w}_i^T\mathbf{q}$ are decorrelated (Hyvärinen and Oja, 2000).

The upper-most layer in Architecture 2 performs linear sparse coding on the SFA outputs. By redefining variables, the ICA layer is denoted by:

$$\mathbf{y}_{\text{ica}}(t) = \mathbf{W}_{\text{ica}}\mathbf{y}_{\text{sfa}}(t), \tag{12}$$

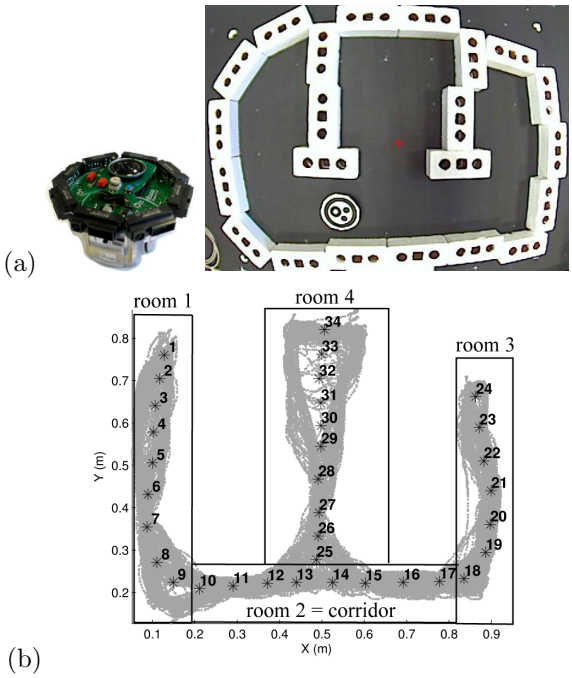


Figure 3: (a) E-puck robot extended with infra-red sensors which can measure distance in the range 4 cm - 30 cm and environment composed of three rooms and one corridor. (b) Robot trajectory in environment for 60.000 timesteps (or 3.3 hours) with labeled asterisks representing delimited locations.

3 Experiments

3.1 Introduction

We use an environment with 3 rooms and a corridor connecting them, shown in Fig. 3.1. There is a camera on top of the environment which records the position of the robot at each timestep. In this way, we can use this information for the supervised learning task (defined in Section 2.1).

Robot and Datasets: The e-puck robot with longer-range [4 cm - 30 cm] infra-red sensors is used for the experiments in this section (see Fig. 3.1). For generating datasets with recorded sensor readings, we use a robot controller written in Matlab that communicates with the e-puck through a Bluetooth link. The robot controller follows a probabilistic wall following algorithm which switches from left to right wall following (or vice-versa) with a certain probability ρ. In this way, the robot trajectory is randomized (see Fig. 3.1). One iteration (for sensing and acting) lasts 200 ms on average.

The speed of the robot is variable as the motor actuators may have values in the interval $\pm$[15,385]. So, the task has to deal with multiple timescales present in the input data. The robot sensors are sequentially read while the robot is moving in the environment, so there might be inconsistencies involved in this process which we do not take into

account because the temporal processing capabilities of the reservoir are supposed to cope with this problem. The signal $\mathbf{u}(t) \in [0,1]$ is built by saving the 8 distance sensors of the robot during navigation and scaling them to the interval $[0,1]$. The total number of samples we recorded amounts to 192.000 timesteps, which means approximately 11 hours of robot navigation.

3.2 Supervised Learning

Previous work has tackled supervised learning with ESNs in localization for simulated mobile robots (Antonelo et al., 2008). Here, we also shown that it perfectly works for real robots. The parameter configuration for this task using Architecture 1 is as follows. The reservoir size is $n_{\text{res}} = 1200$ neurons. The spectral radius is $|\lambda_{max}| = 0.99$. We use three leak rates in the reservoir $\alpha_1 = 0.2$, $\alpha_2 = 0.01$ and $\alpha_3 = 0.005$. The matrix connecting the input to the reservoir ($\mathbf{W}_{\text{in}}$) is initialized to -2, 2 and 0 with probabilities 0.15, 0.15 and 0.7, respectively. The number of outputs in the readout layer corresponds to the number of rooms or locations. At each timestep, there is only one active output found by a winner-take-all function.

The results are shown in Fig. 4. The training process uses 7/8 of the training data (1/8 is used for testing) and it takes around 208 seconds, of which 140 seconds are spent on generating the matrix $\mathbf{W}$ using (1). All experiments are based on a platform with an Intel Core2 Quad CPU and 8 GB memory. The classification performance for room detection is very good considering the random behavior of the robot: 98.1 % of correct classified samples from test data. The upper plot in Fig. 4 shows that the trained RC network is efficient in detecting the current robot room. It is possible to see that most errors are made in the boundaries between a room and the next one, represented by red circles in the figure. When the task is modified to detect the current smaller delimited location (with the same parameter configuration), the trained system can also detect the location with a good performance: 83.3% on test data. These values are the average results over multiple experiments with distinctly generated reservoirs. The standard deviation is small due to the noise added to the sensors (from $N(0, 0.005)$) and the reservoir state update equation for regularization purposes. An analysis of the influence of faulty sensors on the localization ability is planned as future work. In this case, the ESN could easily have additional outputs trained for predicting the value of the faulty sensors for improving robustness.

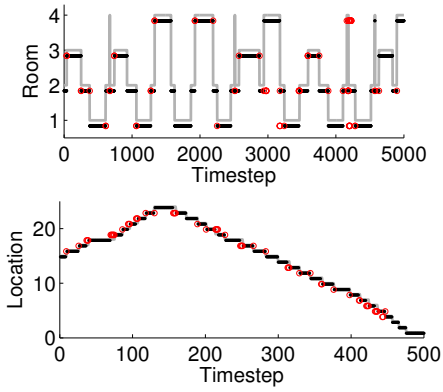


Figure 4: Location detection results on test data with Architecture 1. The grey solid line represents the desired robot location while the real network output is represented by black points (the missclassifications are marked with red circles). Top: the environment is divided in 3 rooms and 1 corridor. Bottom: 34 locations smaller delimited locations as determined by the asterisks in Fig. 3.1

3.3 Unsupervised Learning

We optimized several parameters for Architecture 2 using the error measure defined in Eq. (15):

$$A_j = \frac{\sum_{i \neq p(j)} n(i,j) d(i,p(j))}{\sum_{i \neq p(j)} d(i,p(j))} \quad (13)$$

$$\beta_j = \frac{A_j - n(p(j),j)}{n(p(j),j)} - 0.5\text{kurt}(j) \quad (14)$$

$$B = \sum_j \beta_j / n_{\text{ica}} \quad (15)$$

where: $p(j)$ is the location most representative of ICA unit j (i.e., unit j is most active at location $p(j)$); $n(i,j)$ is the number of samples in which unit j is active (above a certain threshold) for location i; $d(i,k)$ is the distance between locations i and k; $\text{kurt}(j) = \langle (y_{\text{ica}}(j))^4 \rangle - 3$ is the kurtosis for ICA unit j. In this way, β_j is a measure for a place cell j which should be minimized for getting units which only activate for only one specific location (first term in Eq. (14)) and which activates strongly at this location (kurtosis term).

The optimal combination for the number of neurons in each reservoir in the bottom and top RC-SFA modules is $n_{\text{res1}} = n_{\text{res2}} = 600$ neurons. The optimized leak rates were $\alpha_1^{\text{res1}} = 0.1$ $\alpha_2^{\text{res1}} = 0.8$ for the lower reservoir and $\alpha_1^{\text{res2}} = 0.1$ $\alpha_2^{\text{res2}} = 0.05$ for the upper reservoir. The optimal spectral radius in each reservoir was $|\lambda_{max}|_{\text{res1}} = |\lambda_{max}|_{\text{res2}} = 0.99$. Each optimization experiment was executed 10 times. The number of neurons for lower and upper SFA layers, and ICA layer are $n_{\text{sfa1}} = 120$, $n_{\text{sfa2}} = 36$ and $n_{\text{ica}} = 36$. Besides, random noise ($N(0, 0.001)$) is added to the reservoir state update equation (Eq. (1)) during state generation of both reservoirs for the training of upper SFA layers. The matrix connecting the input to the lower reservoir ($\mathbf{W}_{\text{in}}^1$) is initialized to -2, 2 and 0 with probabilities 0.15, 0.15 and 0.7, respectively. For the upper reservoir, $\mathbf{W}_{\text{in}}^2$ is initialized to -0.5, 0.5 and 0 with probabilities 0.15, 0.15 and 0.7, respectively. The

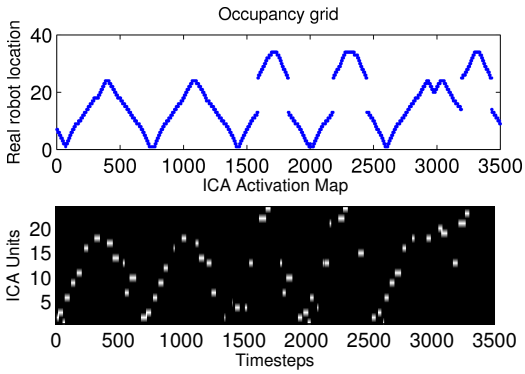


Figure 5: Results using Architecture 2. Top: the real robot occupancy grid. Bottom: the respective spatially-ordered ICA activation map (white dots denote peak responses and black dots represent lower responses).

learning process also occurs in steps, from bottom module to upper modules, taking approximately the same amount of time as in the supervised case.

After applying the learning algorithm, we observe that architecture 2 generates several place cells for the robot environment. In (Antonelo and Schrauwen, 2009), a simpler environment with 2 rooms and a corridor was used to generate place cells, using an architecture with only one RC-SFA module, which could distinguish between two rooms. Although one RC-SFA module can also learn more delimited locations, our architecture with 2 concatenated RC-SFA modules generates qualitatively better place cells, handling multiple timescales present in the input signal. A quantitative analysis and comparison between architectures are left as future work. In Fig. 5, the place cells are ordered so that they correlate with the real robot location. We see that for most of the time there is a place cell which is active. Furthermore, most of the units are dependent on the robot direction, showing that the direction is a slowing-varying feature extracted from the distance sensors as well.

4 Conclusion

In this work, we have shown that it is possible to learn the localization capability for a mobile robot either in a supervised way, showing explicitly the desired location during training, or in an unsupervised way, learning the slow features from the input data to form place cells, using only few distance sensors as input. Our proposed architectures are composed of a randomly generated and fixed recurrent neural network, the reservoir, which functions as a non-linear temporal kernel which expands the input to a high dimensional space. The reservoir states can be linearly combined, using linear regression, to approximate a desired robot location in the output layer or, in an unsupervised way, can be used for non-linear expansion before applying Slow Feature Analysis (SFA) and Independent Component Analysis (ICA) to form place cells.

As future work, we plan to make SFA and ICA an on-line learning algorithm so that the mobile robot can generate place cells in real-time. It is also desired that an on-line reinforcement learning mechanism for autonomous deliberative navigation can be created based on the place cell outputs.

References

Antonelo, E. A. and Schrauwen, B. (2009). Towards autonomous self-localization of small mobile robots using reservoir computing and slow feature analysis, *IEEE Int. Conference on Systems, Man, and Cybernetics*.

Antonelo, E. A., Schrauwen, B. and Stroobandt, D. (2008). Event detection and localization for small mobile robots using reservoir computing, *Neural Networks* **21**: 862–871.

Arleo, A., Smeraldi, F. and Gerstner, W. (2004). Cognitive navigation based on nonuniform gabor space sampling, unsupervised growing networks, and reinforcement learning, *IEEE Trans. on Neural Networks* **15**(3): 639–652.

Franzius, M., Sprekeler, H. and Wiskott, L. (2007). Slowness and sparseness lead to place, head-direction, and spatial-view cells, *PLoS Comput. Biology* **3**(8): 1605–1622.

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications, *Neural Networks* **13**: 411–430.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks, *Technical Report GMD Report 148*, German Nat. Res. Cent. for Inf. Technology.

Milford, M., Schulz, R., Prasser, D., Wyeth, G. and Wiles, J. (2007). Learning spatial concepts from RatSLAM representations, *Robot. Auton. Syst.* **55**(5): 403–410.

Moser, E. I., Kropff, E. and Moser, M.-B. (2008). Place cells, grid cells and the brain's spatial representation system, *Annual Reviews of Neuroscience* **31**: 69–89.

Schrauwen, B., Verstraeten, D. and Van Campenhout, J. (2007). An overview of reservoir computing: theory, applications and implementations, *Proc. of the European Symposium on Artifical Neural Networks (ESANN)*.

Stroesslin, T., Sheynikhovich, D., Chavarriaga, R. and Gerstner, W. (2005). Robust self-localisation and navigation based on hippocampal place cells, *Neural Networks* **18**(9): 1125–1140.

Thrun, S., Burgard, W. and Fox, D. (2005). *Probabilistic Robotics*, The MIT Press.

Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: Unsupervised learning of invariances, *Neural Computation* **14**(4): 715–770.