Generative modeling of autonomous robots and their environments using Reservoir Computing *

Eric A. Antonelo, Benjamin Schrauwen and Jan Van Campenhout
Electronics and Information Systems Department, Ghent University, Belgium
*(*eric.antonelo@elis.ugent.be*)*

Abstract. Autonomous mobile robots form an important research topic in the field of robotics due to their near-term applicability in the real world as domestic service robots. These robots must be designed in an efficient way using training sequences. They need to be aware of their position in the environment and also need to create models of it for deliberative planning. These tasks have to be performed using a limited number of sensors with low accuracy, as well as with a restricted amount of computational power.

In this contribution we show that the recently emerged paradigm of Reservoir Computing (RC) is very well suited to solve all of the above mentioned problems, namely learning by example, robot localization, map and path generation. Reservoir Computing is a technique which enables a system to learn any time-invariant filter of the input by training a simple linear regressor that acts on the states of a high-dimensional but random dynamic system excited by the inputs. In addition, RC is a simple technique featuring ease of training, and low computational and memory demands.

Keywords: reservoir computing, generative modeling, map learning, T-maze task, road sign problem, path generation

1. Introduction

Small and energy efficient autonomous mobile robots form an important research topic in the field of robotics. This is mainly due to their near-term real world applicability as domestic service robots (such as e.g. the small and cheap iRobot service robots). These robots must be easy to *teach* or *program* using training sequences. In order to make them function properly in complex and changing environments, they need to be aware of their position as well as to create models of the environment for deliberative planning. All of this has to be performed using a limited number of sensors with low accuracy, and with a restricted amount of computational power.

This work considers the use of Recurrent Neural Networks (RNN) as a means to solve several problems in mobile robotics. In particular, we are interested in using Reservoir Computing (RC) networks for learning

* This is a draft version. Published in Neural Processing Letters (http://www.springerlink.com/content/72567v2l2j436559/)

robot controllers by example, performing efficient and robust robot localization, and constructing implicit environment maps.

Reservoir Computing has been introduced in [20] as a term which unifies three similar computing techniques: Echo State Networks (ESN) [8], Liquid State Machines (LSM) [12], and BackPropagation DeCorrelation [18]. All three techniques use a *reservoir* of recurrent nodes with fixed and randomly generated weights, while only an output layer (readout) is trained with simple linear regression techniques (see Figure 1). The general applicability and easy of training of RC has been shown in several fields [7, 20, 17].

Robots operating in real indoor environments need to possess robust mechanisms for self-localization so that they can become fully autonomous. This task is considered to be difficult [4]. Traditional algorithms based on the Simultaneous Localization and Mapping (SLAM) concept are expensive to implement due to their limited computational efficiency and also hold uncertainties during the calculation of the robot's pose [4].

Several models in the literature make use of neural networks for map learning in mobile robotics [19]. In [3], a Khepera robot is used to validate a neural network model that learns spatial representations of the environment in the exploration phase (visual stimuli and path integration are the input) while goal-oriented navigation is accomplished by reinforcement learning a posteriori. Other approaches use Self-Organizing Maps for map learning [10, 13]. Several proposals seek to model the place cells found in the hippocampus of rats for implicit map learning and navigation [3]. These cells are activated only when an animal is located at a specific region of the environment.

The robot localization using RC, which we present in this work, forms an implicit model which is based solely on the robot's sensory input history. This setting enables efficient robot localization, although it currently still needs to be trained in a supervised way [2].

We will show not only that RC can be used for localization, but that the same setup can very easily be used to model the robot controller itself, to generate an explicit map of its environment, and even to make long-term prediction of how the controller would react in the environment by both predicting the robot's sensory input as well as the controller's response (or the robot position). The robot, in some sense, can *dream* of how it would react in the future in the current environment. In addition, a higher-order decision system can be built with a scheme where the decision module considers these long-term predictions as relevant input. This can be accomplished by modeling several simple reactive behaviors and generating (dreaming) long-term expectations for each one in order to switch to the most appropriate

behavior at some point in time. This kind of system is also biologically plausible, showing that even simple animals could make long-term predictions about their current actions in the environment.

The innovative nature of this work derives from three main factors: efficient and easy learning of RC networks; the temporal integration of sensory information (by the reservoir which offers a rich pool of dynamics); and the exploitation of the aforementioned characteristics on mobile robot navigation. First, training a RC network is much simpler than training a RNN with standard algorithms (e.g., Backpropagation Through Time) which exhibit slow convergence on training. Second, the short-term memory capabilities of RC networks provide a very important setting for temporal processing of sensory data. Last, building implicit environment configurations with RC networks becomes easier because of its attractive properties.

This paper is organized as follows. Section 2 presents the Echo State Network model used for the experiments in this work as well as the robot model and controller used for generating the training and test datasets. Next, four different experimental setups are worked out (Figure 2). The first setup (Figure 2(a)), elaborated in Section 3, considers the modeling of a robot in a T-maze task by imitation learning (i.e., a RC network is trained to output the robot's actuators given distance and color sensors as input). Next, in Section 4, a RC network is used for robot localization by training it to estimate the robot coordinates and heading given the distance sensors as input to the network (Figure 2(b)). The following section (5) elaborates on effective map learning and generation, which is achieved by letting the resevoir predict the robot's distance sensors given its location as input to the network (Figure 2(c)). The simultaneous generation of paths and environmental perceptions is tackled in Section 6, where a reservoir is trained both ways: to predict the distance sensors as well as the robot's position (Figure 2(d)). Finally the main points of this work and directions for future research are presented in Section 7.

2. Methods

2.1. Reservoir computing

In this work the Echo State Network model [8] is employed for the experiments on the T-maze task, position detection, map learning and path generation. An ESN is composed of a discrete hyperbolic-tangent RNN (i.e., the reservoir) and of a linear readout output layer which maps the reservoir states to the actual output. For the supervised

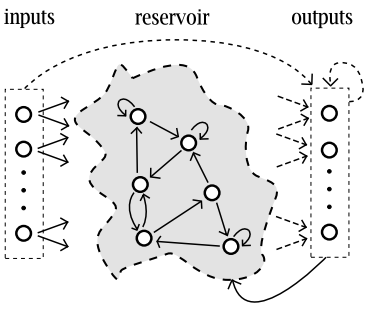


Figure 1. Reservoir Computing network.

training of the ESN, we consider the state update and output equations as follows:

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

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

where $\mathbf{u}$ denotes the input, $\mathbf{x}$ represents the reservoir state, $\mathbf{y}$ is the output and $f() = \tanh()$ is the hyperbolic tangent function. Note that, during training, the output signal is teacher-forced to the reservoir by using the desired output $\mathbf{y_d}$ in the right side of the above equations.

After training, the state-update and output equations become:

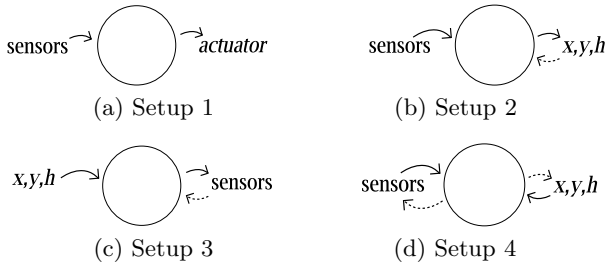
$$\mathbf{x}(t+1) = f\left(W^{\text{res}}_{\text{res}}\mathbf{x}(t) + W^{\text{res}}_{\text{inp}}\mathbf{u}(t) + W^{\text{res}}_{\text{out}}\mathbf{y}(t) + W^{\text{res}}_{\text{bias}}\right) \quad (3)$$

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

where now the actual readout outputs instead of the desired signal are considered in the right hand-side of the equations.

All weight matrices to the reservoir ($W^{\text{res}}_{\cdot}$) are initialised randomly (represented by solid arrows in Figure 1), while all connections to the output ($W^{\text{out}}_{\cdot}$) are trained (represented by dashed arrows in Figure 1). The exact timing of all the different components in these equations is important. The initial state is set to zero. Note that for the various experiments in this work we will not always use all the $W^{\cdot}_{\cdot}$ matrices. For each experiment we will elaborate on the exact setup.

Training is performed using ridge regression, where the regularisation 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 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).



(a) Setup 1 (b) Setup 2

(c) Setup 3 (d) Setup 4

Figure 2. (a) T-maze task modeling. The inputs to the reservoir (circle) are distance and color sensors. The output is the robot actuator (the current direction adjustment). (b) The reservoir is used as a pose detector given the current (distance) sensors as input. (c) Generative setting where the RC network explicity outputs a map of the environment given the actual robot position (x, y) and heading (h) as input. (d) The same reservoir is used for both sensor prediction and pose detection.

Some experiments in this work consider output feedback into the reservoir (i.e., the weight matrix $W_{\text{out}}^{\text{res}}$ is set accordingly). In this case, note that the values which are fed back are the actual output $\mathbf{y}(t)$ calculated by the readout output layer. The output is thus generated in free-run mode.

The Normalized Mean Square Error (NMSE) is used as a performance measure throughout this work and is defined as:

$$\text{NMSE} = \frac{1}{M} \sum_{i=1}^{M} \frac{\langle (y_d^i - y^i)^2 \rangle}{\sigma^2_{y_d^i}} \tag{5}$$

where the sum is over M readout outputs, the numerator of the summand is the mean square error of the i^{th} readout output and the denominator is the variance of the i^{th} desired output.

2.2. Reactive robot controller

The datasets used to train RC networks are generated by a simulator used in [1]. 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 [1]. The robot model is shown in Figure 3. The robot interacts with the environment by distance, color and contact sensors; and by one actuator that controls the movement direction. Sensor positions are distributed uniformly over the front of the robot (from -90° to +90°). Each position holds three sensors (for distance, color and contact perception) [1]. In this work, the robot model has 17 sensor positions. The distance sensors are limited in range and are noisy (they exhibit Gaussian noise on their readings). For distance sensors,

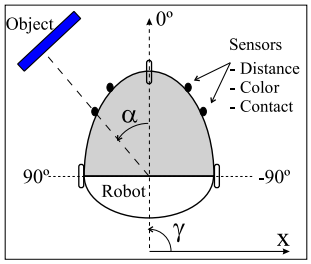


Figure 3. Robot model.

a value of 0 means collision or very close to some object and a value of 1 means the furthest distance from an object. The color sensor results from a normalization of the component Hue of the Hue-Saturation-Value (HSV) color system. The speed 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 [1]) 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 behaviour after training, are used for generating data. Note that the results in this paper are not critically dependent on this specific robot controller. 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 RCToolbox[1] [20].

3. Modeling a controller with long-term memory: the road-sign problem in the T-maze

In this section we will demonstrate that a RC-based system is able to model (learn) a robot controller that needs long-term memory to perform its task. To achieve that we use the simulation setup shown in Figure 2(a), where the reservoir has to drive a robot given its sensory input. No feedback connections to the reservoir are used.

The experiments are performed in a T-maze environment that has a T-shape in which a robot is positioned initially at the end of the longest arm (corridor) (Figure 4). The goal for the robot is located at the upper T-arm located at the same side as an initial sign shown at the beginning of the main corridor. We use the reactive controller

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

to generate trajectories that can be learned by the RC system, but since the reactive controller cannot solve the road-sign problem, we have to help it by placing attractive objects at the beginning of the corridors it has to turn into (marked by crosses in Figure 4). These attract the robot to the correct goal (so the reactive controller does not solve the T-maze task). These objects are not visible to the RC system. The reservoir thus has to learn the long-term temporal dependencies between relevant events (i.e., the sign in the corridor and the decision at the T-junction) which are demonstrated by the recorded trajectories. We consider distinct sizes for the T-maze in order to test the memory capacity of RC networks (see Figure 4).

The reservoir configuration in this section is as follows. The inputs to the network are distance and color sensors totalling 34 inputs which can range from 0 to 1 (for distance sensors, 0 means near and 1 means far - see previous section). The reservoir is composed of 400 sigmoidal nodes, scaled to a spectral radius [2] of $|\lambda_{max}| = 0.9$ [6], which approximately sets the reservoir at the edge of stability. The readout layer has 1 output unit which corresponds to the robot actuator (i.e., the direction adjustment, as the robot has constant speed). The weight matrix from input to the reservoir is initialized at the values -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 (as you will see, a different configuration is used for other experiments).

The original dataset collected from the simulator is downsampled[3] by a factor of 100, which is equivalent to slowing down the reservoir time scale [16, 9]. This is because the robot has a relatively constant low velocity, taking about 1300 timesteps to go from the start position to the goal in environment E1 (Figure 4).

The datasets (sensory inputs and motor output) collected from the simulator are divided in three parts, where two parts are used for training and the other one for testing. The current stage of the work does not include the test of the RC network as a controller embedded in the simulator. It is only tested on pre-recorded sensory inputs from the simulator (not real-time). The training is performed based on a 3-fold cross validation with ridge regression as the learning algorithm. We add a large amount of noise on distance sensors (Gaussian noise from $N(0, 0.2)$ which means effectively $N(0, 60)$ in distance units) and on color sensors (Gaussian noise from $N(0, 0.1)$) before resampling the

[2] The spectral radious λ_{max} is an eigenvalue of the matrix $W_{\text{res}}^{\text{res}}$ with the largest absolute value.

[3] The downsampling of the data is made with the Matlab *resample* function unless stated otherwise

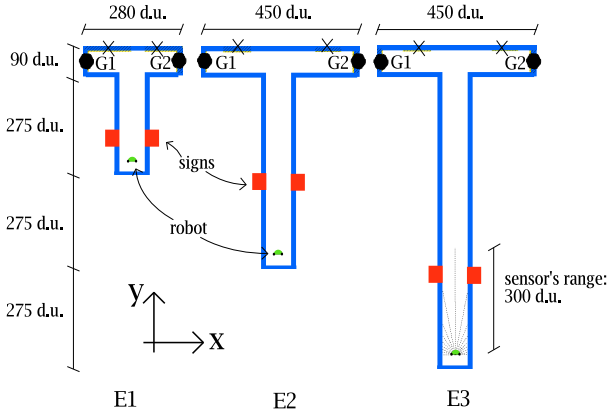


Figure 4. T-maze environments. Dimensions are given in *distance units* (d.u.). In the T-maze task, a sign at the left (right) indicates that the goal is at the left arm, in G1 (right arm, in G2). Crosses mark the positions of attractive objects which are visible to the reactive robot controller (which automates the generation of the training samples) but not for the RC network. Note that both 2 signs are shown in the figure for visualization purposes (the robot only *sees* 1 sign).

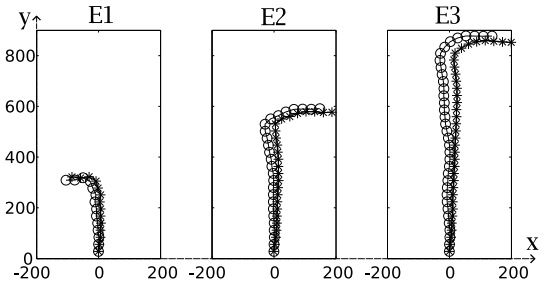


Figure 5. Predicted robot paths. Each point of the desired trajectory is marked by an asterisk while circles represent points of the predicted robot path. From left to right are the plots for T-mazes E1, E2 and E3 respectively.

dataset (which are high noise rates when compared to [15] who consider noise-free data).

Results are summarised in Table I. Each experiment has a dataset with 60 examples (each example is built with sensor and actuator samples from a robot going from the start position to the goal position). Each experiment is evaluated 30 times (runs) with different stochastically generated reservoirs and the results (error) are averaged over these 30 runs.

Remember that the output of the RC system is the actuator of the robot. To evaluate whether the predicted trajectory is *correct* (i.e. whether the reservoir is able to drive the robot through the T-maze and solve the task correctly), we plot the real trajectory and the predicted

Table I. T-maze task results[A].

Environment	Training NMSE		Test NMSE		No. Effective Trajectories	
	Mean	Std	Mean	Std	Training	Test
E1	0.0144	0.0009	0.0220	0.0015	40/40	20/20
E2	0.0338	0.0021	0.0638	0.0041	40/40	19/20
E3	0.2074	0.0410	0.3122	0.0509	20/40	8/20

A. For the fourth column, 40 trajectories is the maximum value of effective trajectories for the training set and 20 for the test set.

trajectory by moving two points in X and Y coordinates as they were driven by the desired and predicted actuators, respectively. Figure 5 shows three plots for T-mazes E1, E2 and E3. They show the real trajectory of the robot (each point given by an asterisk) and the trajectory formed by reading the reservoir responses (each point represented by circles). Note that the predicted trajectory was built by stimulating the reservoir with test data not contained in the training data.

It is easily seen that as the size of the corridor increases the difficulty for getting a 100% effective reservoir is higher. For T-mazes E1 and E2, the test error is very low (Table I), while for the longer T-maze E3, the error significantly increases. This demonstrates that the reservoir's memory is limited, and that, as the corridors get longer, it is more difficult to remember the initial signal. This can also be seen through visual inspection of the reservoir's performance (i.e., comparing the real and predicted trajectories). This is done by training a reservoir network on a number of examples, and counting the number of effective trajectories for the training and test sets (whose size are 40 and 20 examples respectively). The results are also shown in Table I. Note that we would expect increase in performance if the experiments were accomplished in real-time (i.e., the RC network embedded in the simulator would drive the robot in real-time) because the distance sensors would reflect the environmental perception changes caused by the actuator which is controlled by the RC network.

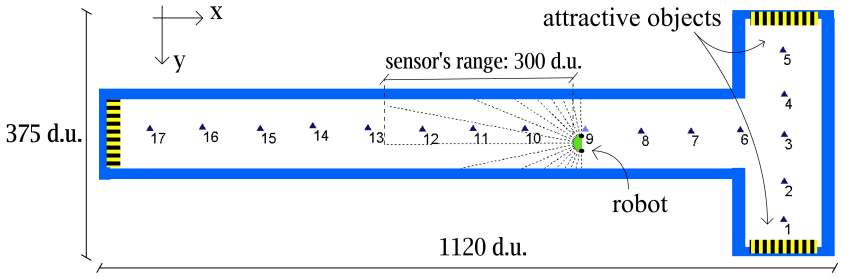
The memory of the reservoir can become greater by increasing the resampling rate[16], or by slowing down the reservoir dynamics (which is equivalent to adding leaky integrators) [9]. However, there are limitations for slowing down the dynamics since the reservoir still needs to be fast enough to generate the turning movement into the narrow corridors.

4. Position detection

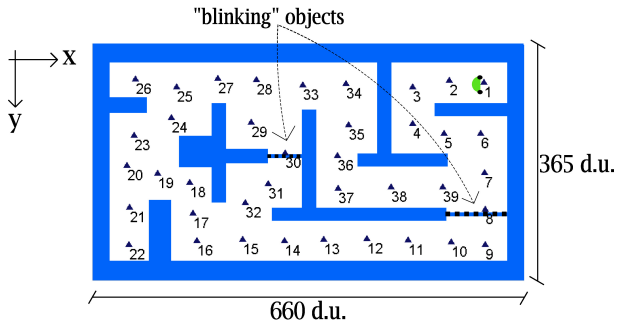
Robot localization is a pre-requisite for making a robot fully autonomous in indoor environments. Localization usually requires the building of an explicit map of the environment which is then used to improve the estimation of the robot pose given by dead reckoning. In this section we show that it is possible to use a RC network as a position detector by only considering the history of the robot's sensory inputs. This setup was first demonstrated in [2] where by labelling specific locations in the environment, the RC system was able to perform localization in the form of a classification task (this is also achieved with Long Short-Term Memory networks in [5]). This section shows that a RC network can also perform position estimation in the form of a regression task where the exact coordinates are expected as output. We use the RC setup shown in Figure 2(b) where the estimated positions are fed back into the reservoir. This additional stimulation endows the reservoir with long-term memory [11] for the position estimation task effectively improving its performance (the fading memory of a reservoir without feedback connections from the readout layer is not enough for satisfactory performance).

The environments used for the current task are shown in Figure 6. The first one is a long T-maze environment which is used to test the memory capacity of the RC network. When the robot is driving along the longest corridor, the frontal distance sensors do not detect the ending corner of the corridor which is out of range. Therefore, for correct position detection, it is strictly necessary that some sort of short-term memory is present in the reservoir.

The simulations for the generation of trajectories are explained in the following. The robot trajectory in environment E4 depends on the current visible target - only a (randomly chosen) attractive object is visible at a time. The controller keeps on capturing targets indefinitely during simulation (a target is hidden when captured and another one is made visible). In environment E5, the robot's trajectory depends on the dynamics of the blinking objects (see Figure 6) and Gaussian noise is added to the motor output, influencing the trajectory as well. This diversification of trajectories is accomplished in order to make it more difficult for the prediction problem while it is not critical for the results. The RC network is configured in the following way for experiments with environment E4. The inputs to the network are the distance sensors. The reservoir has 400 nodes, whose connection matrix is scaled to a spectral radius of $|\lambda_{max}| = 0.9$ [6]. The readout layer has 3 output units which corresponds to the normalised robot coordinates and heading. The weight matrices from inputs and outputs to the



(a) E4



(b) E5

Figure 6. Environments used for position detection and map learning experiments. Dimensions are given in *distance units* (d.u.). (a) Long T-maze environment. Locations are marked by small labeled triangles and attractive objects are located at each arm's corner. (b) Complex room environment with two dynamic objects at locations 8 and 30 which blinks by random time intervals (blocking or freeing the robot's way).

reservoir are initialized at the values -0.4, 0.4 and 0 with probabilities 0.2, 0.2 and 0.6, respectively.

Both ridge regression for the output training and the adding of Gaussian noise with variance of 0.001 to the state update equation are used for regularisation. Only using ridge regression is not enough to get stable output feedback behaviour. The original dataset from the simulator is downsampled by a factor of 30. The resulting dataset is divided in two parts of 2400 and 1200 timesteps which are used for training and testing, respectively.

The differently configured parameters for experiments with environment E5 are presented next. The reservoir size is 800 neurons. The weight matrices from inputs (outputs) to the reservoir are initialized at the values -0.6, 0.6 and 0 (-0.025, 0.025 and 0) with probabilities 0.25, 0.25 and 0.5 (0.15, 0.15 and 0.7), respectively. The dataset is downsampled by a factor of 50. The training and test datasets consist of 5600 and 800 timesteps after resampling, respectively.

The results on test data can be seen in Figure 8. The first plot shows that the RC system can cope very well with the long T-maze. The robot trajectory is not completely repetitive: between timesteps 200 and 400, the robot captures two targets (see the changes on Y coordinate) whereas between timesteps 800 and 1200, four targets are captured (the Y robot coordinate fluctuates for a longer period). The average NMSE over several generated reservoirs (30 runs) is 0.025 with standard deviation of 0.02.

The second plot shows the performance on test data for environment E5. This second experiment required more parameter tuning than the first one due to the apparent increased complexity associated with the robot trajectory (see Figure 8). Nonetheless, the RC network correctly follows the target robot coordinates (x,y and heading). The average NMSE over 30 runs is 0.283 and the standard deviation is 0.072.

Consider that the task of robot localization has to be accomplished even in the presence of faulty sensors. If these broken sensors can be detected and then predicted with some technique, we obtain a fault-tolerant robot localization system. Consider Setup 4 now (Figure 2(d)). By creating connections from the same reservoir to the sensors and letting them to be output nodes which can be trained, we establish a way of using the same reservoir for sensor prediction and robot localization. In Figure 7, we can compare the results from experiments using Setup 2 (dashed line) and Setup 4 (solid line). We consider the task of position estimation in environment E5 (complex room) and the same training and test datasets. Each experiment is executed 20 times considering that in the test dataset randomly chosen sensors are broken (by setting their outputs to zero). As the number of broken sensors increases, the RC network configured with Setup 2 deteriorates notably but gracefully in performance. On the other hand, the generative setting which predicts the broken sensors can cope very well with the faulty sensors (as opposed to SLAM, RC-based localization can deal with broken sensors and does not need expensive laser scanners, although currently we still need supervised learning to achieve that).

These experiments show that RC is able to create an implicit map of the environment (while SLAM requires explicit maps to be stored) and use it for localization purposes, both on a coordinate basis and on a more abstract location basis (as previously showed in [2]). It works in environments that look spatially very similar and need longer-term memory to know the correct location, and even in complex maze-type environments. This RC-based solution for robot localization is established completely on a black-box approach whereas it is very easy to train the network. In future work we plan to use the output of this

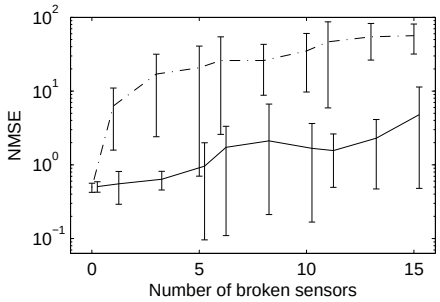


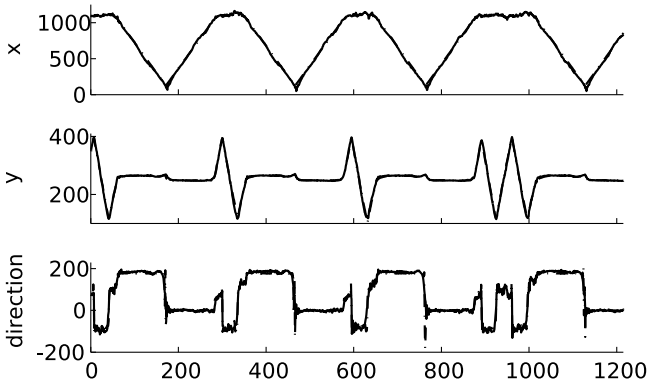
Figure 7. Results achieved by letting the sensors to be faulty. The dashed line is the average error using Setup 2 whereas the solid line is the error using the Setup 4 (by predicting the broken sensors).

system as correcting input for a Kalman filter based dead reckoning system.

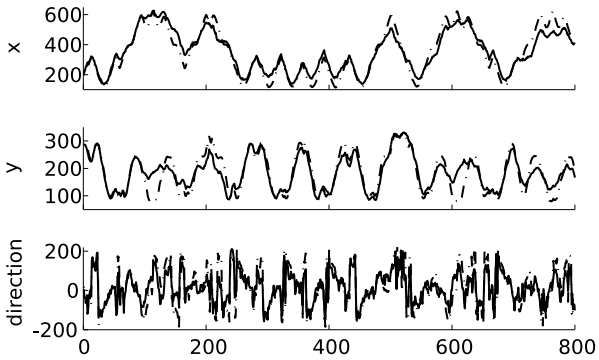
5. Map learning

In the above position detection task, a RC network is used to predict the robot position given distance sensors as input. It thus constructs an implicit map of the environment that is used for localization. Here we consider the reverse problem, that is, given the robot position as input, the reservoir has to predict the expected sensory input (Figure 2(c)). In this way, by driving the robot in the environment and recording its position and heading sequence, the RC network can be trained for generating a map of the environment. The implicit map from the previous section can thus be made explicit. We will similarly show that we can also generate the maps from stream of more higher order location. The locations are defined by the labelled triangles in Figure 6. The robot is in a location when it is closest to the given triangle. No heading input is provided in this case. We consider the same environments as in the previous section (Figure 6) and the same experimental setup for the parameters. The only difference in this section is that the inputs to the reservoir are the (normalised) robot coordinates and heading whereas the readout output layer (which is fed back into the reservoir) is composed of 17 output nodes, corresponding to the distance sensors. Additionaly, the downsampling used for the map learning experiments is made through decimation (i.e. taking 1 sample every n timesteps) so that corners are well represented (otherwise the map is inconsistent). We will also look at the influence of adding color information as input.

The maps are built by moving the robot according to the pre-recorded test data from the simulator and plotting the distance sensor



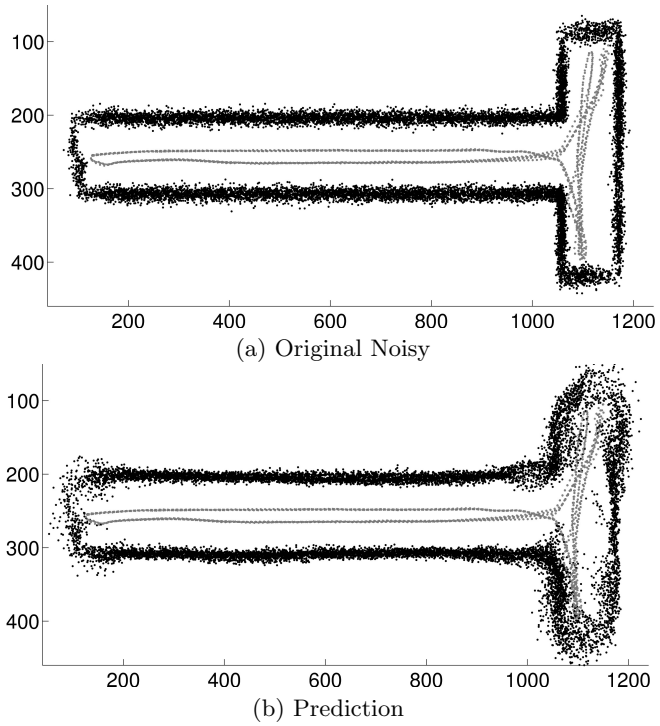
(a) Position estimation for E4



(b) Position estimation for E5

Figure 8. Position detection by the RC network. The dashed lines represent the target position (robot's coordinates and heading) whereas the solid line is the output of the RC network on test data. (a): plot for environment E4 (long T-maze) (dashed lines not visible). (b): plot for environment E5 (complex room).

readings (or predictions) from the robot's local coordinate system (see Figure 9). The maps for the test data are generated by running the RC network for 1300 and 2000 timesteps for environments E4 and E5, respectively. For the long T-maze environment, the generated map (Figure 9(b)) is very similar to the real map. Good performance is also achieved for the more complex maze environment (Figure 10) considering either the robot coordinates or the locations as input (the maps are very similar). The more abstract concept of location is represented by a binary vector. The map generation is made by stimulating the reservoir with the robot coordinates and heading (or locations). The robot trajectory follows a dynamics which is probably used by the reservoir for map generation. In order to find out how the trajectory

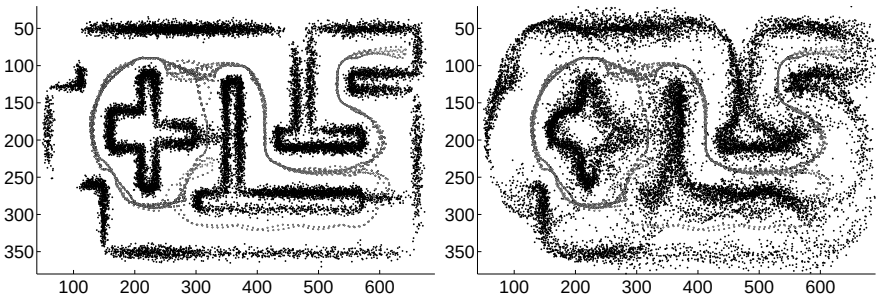


(a) Original Noisy

(b) Prediction

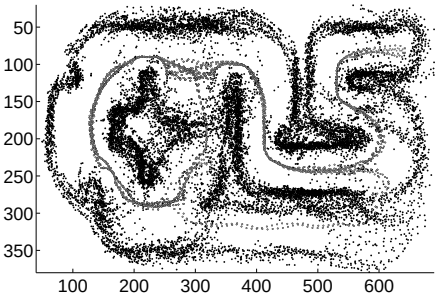
Figure 9. Original and predicted maps for long T-maze (E4). Black points represent the sensor readings whereas gray points are the robot position. (a): the real noisy map as seen by the robot. (b): the map generated by the RC network (after training) given the robot coordinates and heading as input.

dynamics is related to reservoir performance, a new experiment was setup with environment E4 by placing four additional attractive objects at the longest corridor. The dashed arrows in Figure 11(a) indicate the positions of attractive objects. Gaussian noise is also added to motor output. Thus the robot trajectory is diversified with these new changes. The map in Figure 11(a) is built with 2800 timesteps data. The noise version of the map used for training (Figure 11(b)) and corresponding predictions (Figure 11(c)(d)) are constructed with 750 timesteps data. If the reservoir only considers distance sensor as input (but no color sensor data), the irregular trajectories in the long corridor cause the incorrect prediction in the form of displaced wall segments (Figure 11(d)). By considering the additional color information, the RC-based map generation is much improved and no shifted walls are present (Figure 11(c)). Therefore, it is necessary that extra environment information (like color sensor data) is included as input to the reservoir in order to cope with complex trajectories in the map generation task. Note that the process of including new data into the prediction model is



(a) Original Noisy

(b) Prediction from robot coordinates and heading



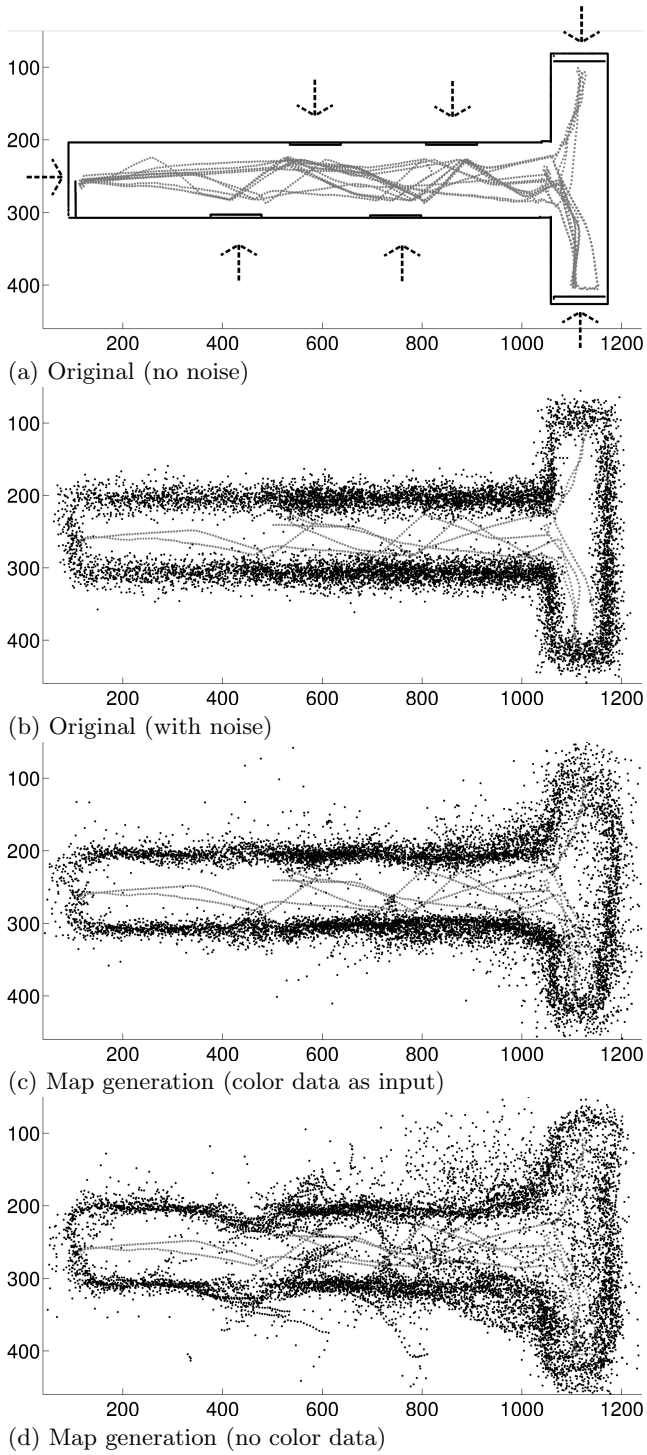
(b) Prediction from locations

Figure 10. Real and predicted maps for complex room (E5). Black points represent the sensor readings whereas gray points are the robot trajectory. (a): the real noisy map. (b): the map generated by the RC network trained with the map in (a) (the inputs to the reservoir are the robot coordinates and heading). (c) the generated map with the more abstract locations as input.

straightforward, requiring just the addition of more inputs (the model is automatically adjusted by a general reservoir and a general learning algorithm).

6. Path generation

We have shown that we can use RC networks for both position detection and map generation with the same reservoir configuration. This section presents an interesting setting where the same reservoir can be used for both aforementioned tasks simultaneously (Figure 2(d)). Now the reservoir only has outputs which are fed back to the reservoir by either teacher-forcing or using it in the free-run mode (i.e., the real network output is fed back). The output nodes are the robot position as well as the robot sensor readings. If we want to predict the robot sensors, we can teacher-force the robot position. Accordingly, we can teacher-force the sensor readings for position estimation.



(a) Original (no noise)

(b) Original (with noise)

(c) Map generation (color data as input)

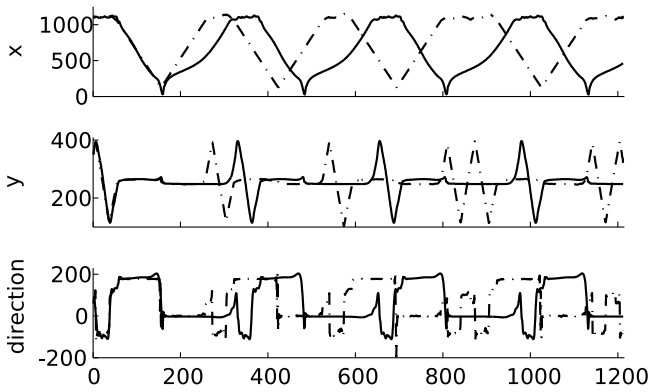
(d) Map generation (no color data)

Figure 11. Map generation from more irregular robot trajectories for the long T–maze (E5). (a): the original map. The dashed arrows show the position of attractive objects. (b): the map used for training (noise added from $N(0, 9)$). (c): the map generated considering robot coordinates, heading and color sensors as input to the reservoir. (d): the map generated excluding color sensor data.

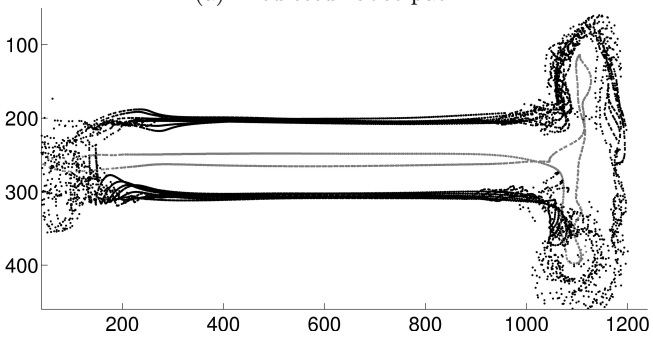
The RC network is configured in the following way for the experiments in this section. The readout layer has 20 output nodes (17 for distance sensors plus 3 for normalised robot coordinates and heading). The reservoir is made of 700 neurons, with a spectral radius $|\lambda_{max}| = 0.9$. The values in the weight connection matrix from output to reservoir are initialized to -0.5, 0.5, and 0 with probabilities 0.2, 0.2, and 0.6, respectively. The noise in the state update equation is 0.0001.

Consider environment E4 (the long T-maze). The robot navigates continuously through this environment and its sensors and position are recorded. The resulting dataset (the same as previous section) contains 3600 timesteps after resampling. The RC network is trained with the first 2400 timesteps. Then, we set the initial state of the reservoir to the state it was at timestep 2400 during training. After that, the reservoir runs in free-mode, where every actual output is fedback to the reservoir (this can be thought as follows: the robot runs for 2400 timesteps and then it *dreams* of its trajectory and its environment in the following timesteps). As the output is not teacher-forced, the reservoir is free to develop its own dynamics. The results are shown in Figure 12. The first plot shows the predicted robot position whereas the second one corresponds to the generated map. As it can be seen, the predicted robot trajectory follows a cyclic dynamics as if the robot was capturing each target in a predefined sequence. Probably this sequence is the most relevant in the training dataset. The generated map shows that the long corridor is well rebuilt while the corners are more difficult to reconstruct due to the fast turning robot behavior at these locations.

This work demonstrates that a single RC system is able to both model the robot's controller and environment as well as to generate trajectories and environments in a free-run fashion. These capabilities are processed by a single recurrent neural network without any rule-based mechanism or higher-order technique. On the other hand, this neural network model could be used by some more abstract decision-based system, for instance: for making future predictions on the robot's path and environment if a certain behaviour is chosen from some point on (in this case a behavior module would be input to a RC network). Although the term might seem quite colloquial this can be understood as if the robot was dreaming about its environment and its associated reactive behaviour. Furthermore, this can be achieved with a system that is biologically plausible [21] which is an evidence that even very simple animals could make short-term predictions of their actions in their environment.



(a) Predicted robot path



(b) Generated map

Figure 12. Path generation in free-run mode. The RC network predicts both robot position and the sensor readings (there is no teacher forcing during prediction). (a): the predicted robot path. (b): the corresponding generated map (black points are the sensor readings and gray is the predicted robot trajectory).

7. Conclusions

The Reservoir Computing (RC) approach is a new concept for efficient and easy training of Recurrent Neural Networks (RNNs). With RC, the states of a random dynamical system made of a fixed RNN are mapped onto the readout output layer. Only this readout layer is trained with standard linear regression techniques, that is, training consists of computing the transpose of matrices and matrix inversion operations. RC is a real alternative to previous techniques such as Backpropagation Through Time (BPTT) which shows slow convergence and very difficult training for RNNs.

In this work we have shown that RC can be used to model and predict several aspects of autonomous mobile robots. First, we used RC for training robot controllers to solve the T-maze task, which requires long-term memory in order to establish the association between

a stimulus in the past and the correct action at a specific point in the future. Next RC was used for enabling efficient and robust robot localization even when some distance sensors are faulty. The localization was possible with few (17) sensors as opposed to SLAM which usually requires expensive laser range scanners. The following applications of RC were in map and path generation, where a reservoir was used in a generative setting for making explicit the map stored internally by the RC network. All of these applications are based on implicit models (no explicit representation of the environment is used). In addition, the use of output feedback connections in RC improves the estimation of positions (in the localization task) and sensory inputs (in the map generation task), while in the path generation task it enables long-term future predictions of both the environment and the associated robot's reactive behavior.

RC has biological foundations as it is shown, for example, that Liquid State Machines (a type of RC) are based on the micro-column structure in the cortex [12]. Furthermore, the works in [14, 21] establish a strong association between real brains and reservoirs, what is an appealing motivation towards the use of RC for intelligent autonomous systems.

As future work we plan to validate the above experiments in a real-world robotic setup. Additionaly, one of the most relevant points to be investigated relies on the storage capacity of RC-based systems. The following question is pertinent: How large can the environments be so that they can still be modeled? Current research indicates that the storage capacity could be increased by elaborating hierarchical reservoirs. Thus, specific parts of the environment would be stored by distinct reservoirs.

In this work the tranining of RC networks has been done off-line and in a supervised way. An interesting future direction for research is the elaboration of a system which can generate and learn new locations in an unsupervised way and as such autonomously construct place cell representations.

Acknowledgements

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

References

1. E. A. Antonelo, A.-J. Baerlvedt, T. Rognvaldsson, and M. Figueiredo. Modular neural network and classical reinforcement learning for autonomous robot navigation: Inhibiting undesirable behaviors. In *Proceedings of IJCNN 2006*, Vancouver, Canada, 2006.
2. E. A. Antonelo, B. Schrauwen, X. Dutoit, D. Stroobandt, and M. Nuttin. Event detection and localization in mobile robot navigation using reservoir computing. In J. M. de Sa et al., editor, *ICANN, Part II*, pages 660–669. Springer-Verlag Berlin Heidelberg, 2007.
3. A. Arleo, F. Smeraldi, and W. Gerstner. Cognitive navigation based on nonuniform gabor space sampling, unsupervised growing networks, and reinforcement learning. *IEEE Transactions on NN*, 15(3):639–652, May 2004.
4. T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part ii state of the art. *Robotics and Automation Magazine*, September 2006.
5. A. Forster, A. Graves, and J. Schmidhuber. RNN-based learning of compact maps for efficient robot localization. In *Proceedings of ESANN*, 2007.
6. H. Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, 2001.
7. H. Jaeger. Tutorial on training recurrent neural networks, covering bptt, rtrl, ekf and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology, 2002.
8. H. Jaeger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 308:78–80, April 2 2004.
9. H. Jaeger, M. Lukosevicius, and D. Popovici. Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20:335–352, 2007.
10. B. J. A. Krose and M. Eecen. A self-organizing representation of sensor space for mobile robot navigation. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, September 1994.
11. W. Maass, P. Joshi, and E. Sontag. Principles of real-time computing with feedback applied to cortical microcircuit models. In *Advances in Neural Information Processing Systems*, volume 18, 2006.
12. W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.
13. S. Najand, Z. Lo, and B. Bavarian. Applications of self-organizing neural networks for mobile robot environment learning. In G. A. Bekey and K. Y. Goldberg, editors, *Neural networks in robotics*, pages 85–96. Kluwer Academic Publishers, 1992.
14. D. Nikolić, S. Häusler, W. Singer, and W. Maass. Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Advances in Neural Information Processing Systems 19*, volume 19, 2007. in press.
15. R. M. Rylatt and C. A. Czarnecki. Embedding connectionist autonomous agents in time: The 'road sign problem'. *Neural Processing Letters*, 12:145–158, 2000.

16. B. Schrauwen, M. D'Haene, D. Verstraeten, and J. Van Campenhout. Compact hardware for real-time speech recognition using a liquid state machine. In *Proceedings of the IJCNN*, 2007. submitted.
17. B. Schrauwen, D. Verstraeten, and J. Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the European Symposium on Artifical Neural Networks (ESANN)*, 2007.
18. J. J. Steil. Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity. In *Proceedings of IJCNN '04*, volume 1, pages 843–848, 2004.
19. S. Thrun. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 99:21–71, 1998.
20. D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. A unifying comparison of reservoir computing methods. *Neural Networks*, 20:391–403, 2007.
21. T. Yamazaki and S. Tanaka. The cerebellum as a liquid state machine. *Neural Networks*, 20:290–297, 2007.

Address for Offprints: Electronics and Information Systems (ELIS) department Ghent University Sint Pietersnieuwstraat 41 9000 Ghent Belgium