# Modeling Multiple Autonomous Robot Behaviors and Behavior Switching with a Single Reservoir Computing Network

Eric Antonelo, Benjamin Schrauwen, Dirk Stroobandt
Department of Electronics and Information Systems
Ghent University
Ghent, Belgium
eric.antonelo@elis.ugent.be

***Abstract*—Reservoir Computing (RC) uses a randomly created Recurrent Neural Network as a reservoir of rich dynamics which projects the input to a high dimensional space. These projections are mapped to the desired output using a linear output layer, which is the only part being trained by standard linear regression. In this work, RC is used for imitation learning of multiple behaviors which are generated by different controllers using an intelligent navigation system for mobile robots previously published in literature. Target seeking and exploration behaviors are conflicting behaviors which are modeled with a single RC network. The switching between the learned behaviors is implemented by an extra input which is able to change the dynamics of the reservoir, and in this way, change the behavior of the system. Experiments show the capabilities of Reservoir Computing for modeling multiple behaviors and behavior switching.**

***Index Terms*—reservoir computing, autonomous robot navigation, imitation learning, behavior switching.**

## I. Introduction

Autonomous mobile robots are becoming increasingly important in our society. Service robotics is the current promising area after the industrial robotics era. In this sense, it would be very elegant to have a technique which can teach a robot to accomplish tasks just by imitation. In this case, the human user could show to the robot how to accomplish tasks, for instance, by giving examples of movements, behaviors or trajectories to be followed. In this sense, a mobile robot should learn and generalize what it has learned by imitation. Furthermore, the robot needs to learn to accomplish more than one task or generate multiple behaviors. It also must be able to switch between them, depending on the current (possibly human) need.

This work seeks to answer the aforementioned questions, and investigates how a single Recurrent Neural Network (RNN) could solve the problem. Our approach is based on Reservoir Computing (RC) [1]. Reservoir Computing uses a fixed (usually random) RNN that is used as a reservoir of rich dynamics, and a linear static readout output layer (see Fig. 1). Only the output layer is trained in a supervised way, while the recurrent part of the network (the so called *reservoir*) has fixed weights. The reservoir weights are usually scaled so that the network's dynamic regime is situated at the edge of stability. Reservoir computing is a unifying term for three computing techniques, namely, Echo State Networks [2], Liquid State Machines [3], and BackPropagation DeCorrelation [4]. Theoretical analysis of reservoir computing methods [5] and a broad range of applications [1], [6] (which sometimes even drastically outperform the current state-of-the-art [7]) 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.

Reservoir Computing has been successfully applied to a wide range of robotic tasks. In [8], RC is used for complex event detection and robot localization in the context of small mobile robots with just a few noisy sensors. In that work, two different robot models are used, including the e-puck robot [9] with 8 infra-red sensors. In [10], RC is used in various robotic tasks including prediction of robot coordinates, map learning and path generation. RC has also been used for modeling the road sign problem in [11], where a mobile robot must remember a previously given stimulus (light sign) in order to accomplish a delayed-response task successfully. All these robotic tasks are performed proficiently by a RC network. The short-term memory in the reservoir enables more complex computation that would not be possible otherwise, while the training algorithm simply adjusts output weights by using linear regression methods.

In [12], a hierarchical neural network (with two RNNs situated in different levels) learn by back-propagation through time (BPTT) [13] to generate sequences of behavior patterns by imitation learning of a robotic arm. Complex dynamics and training in such RNNs hinder the modeling task as well as limit the applicability of the proposed approaches.

The current work uses Reservoir Computing as an alternative method for modeling multiple autonomous robot behaviors by imitation learning. The simple training of such networks (compared to BPTT) and its short-term memory capabilities are characteristics which greatly help this modeling task.

In previous work [14], an intelligent navigation system for mobile robots [15] is identified by a RC network (learning by examples). After training, the RC network is able to imitate the original controller with an increase in stability, by reproducing

obstacle avoidance and target seeking behaviors. However, it is not modeled to simultaneously learn different teacher controllers generating conflicting behaviors (as we will study in this work). These teacher robot controllers will be constructed using different versions of an intelligent navigation system for mobile robots [15]. Each teacher controller provides examples of navigation strategies which will be used for training a single RC network. The first teacher controller performs exploration of the environment ignoring the existent targets whereas the second teacher controller seeks and captures targets in the environment, avoiding collision with obstacles. These two behaviors are conflicting behaviors, generated by distinct teacher controllers, which will be learned by a single RC network. The resulting RC-based robot controller will be able to navigate in the environment and accomplish tasks according to the currently selected behavior. The conflicting behaviors are represented in a distributed way in the network. Furthermore, the behavior switching is simply modeled as an extra input to the network, which tells what behavior should be selected.

This paper will show that Reservoir Computing is a promising technique in robotics, that can be readily applied to multiple behavior modeling and presents a reliable and easy way of switching between learned behaviors.

## II. Reservoir computing

The RC network model used in this work follows the Echo State Network (ESN) approach [2]. 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 general state update equation for the nodes in the reservoir and the readout output equation are as follows:

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

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

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 (most common type of activation function used for ESNs). The weight matrices $\mathbf{W}$ represent the connections between the nodes of the network (where $r, i, o, b$ denotes $reservoir, input, output,$ and $bias$, respectively). All weight matrices to the reservoir (denoted as $\mathbf{W}_{\cdot}^{\mathrm{r}}$) are initialized randomly (represented by solid arrows in Fig. 1), while all connections to the output (denoted as $\mathbf{W}_{\cdot}^{\mathrm{o}}$) are trained (represented by dashed arrows in Fig. 1). The initial state is set to $\mathbf{x}(0) = \mathbf{0}$.

However, we do not use the output feedback to the reservoir because the problems in this work do not require a very long-term memory:

$$\mathbf{x}(t+1) \quad = \quad f(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}(t) + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}(t) + \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}). \quad (3)$$

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

$$\mathbf{y}(t+1) = \mathbf{W}_{\mathrm{r}}^{\mathrm{o}}\mathbf{x}(t+1) + \mathbf{W}_{\mathrm{b}}^{\mathrm{o}}. \quad (4)$$

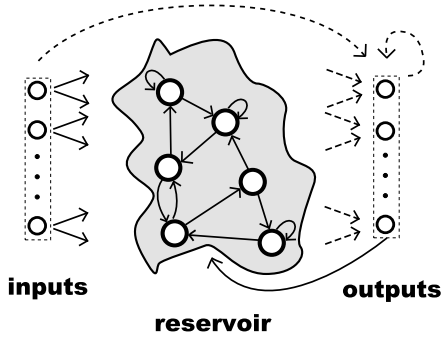



Fig. 1. Reservoir Computing network. The reservoir is a dynamical system of recurrent nodes. Solid lines represent connections which are fixed. Dashed lines are the connections which can be trained.

Each element of the connection matrix $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ is drawn from a normal distribution with mean 0 and variance 1. The randomly created $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ matrix is rescaled such that the system is stable and the reservoir has the echo state property (i.e., it has a fading memory [5]). This can be accomplished by rescaling the matrix so that the spectral radius $|\lambda_{max}|$ (the largest absolute eigenvalue) of the linearized system is smaller than one [5]. Standard settings of $|\lambda_{max}|$ lie in a range between $0.7$ and $0.98$ [16]. In this work we scale all reservoirs ($\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$) to a spectral radius of $|\lambda_{max}| = 0.9$ which is an arbitrarily chosen value (the optimization of the spectral radius for each experiment was not necessary because the changes in performance were not very significant).

An ESN without output feedback is inherently stable due to the echo state property [5]. However, with nonzero output feedback, stability can not be always guaranteed. A formal analysis of the stability of the ESN in this case is challenging. Nevertheless, stabilizing solutions include the use of regularization techniques such as the addition of state noise during training [16].

Next, consider the following notation: $n_i$ is the number of inputs; $n_r$ is the number of neurons in the reservoir; $n_o$ is the number of outputs.

The imitation learning process uses training data from two teacher controllers. Consider that the data (robot sensors and actuators) obtained from both controllers are concatenated into a single dataset and that the total number of time samples is $n_s$. Training is performed using linear regression (least squares) on 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 (3). In this process, state noise can be added to (3) for regularization purposes, although the current task does not need it because the sensors and actuators from the teacher robot controllers are already noisy. 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 actuators) are collected row-wise into a matrix $\hat{\mathbf{Y}}$. Then, the readout output's matrix $\mathbf{W}_{\mathrm{rb}}^{\mathrm{o}}$ (i.e., the column-wise concatenation of $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$ and $\mathbf{W}_{\mathrm{b}}^{\mathrm{o}}$) of size $(n_r + 1) \times n_o$ is

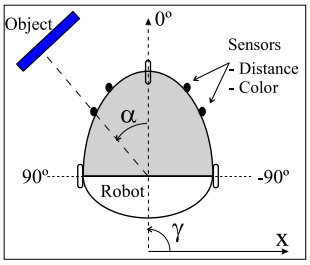



Fig. 2. Robot model from SINAR simulator.

created by solving (in the mean square sense):

$$\mathbf{M}\mathbf{W}_{\mathrm{rb}}^{\mathrm{o}} = \hat{\mathbf{Y}} \quad (5)$$

$$\mathbf{W}_{\mathrm{rb}}^{\mathrm{o}} = (\mathbf{M}^{\top}\mathbf{M})^{-1}\mathbf{M}^{\top}\hat{\mathbf{Y}} \quad (6)$$

Note that the other matrices ($\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}, \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}, \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}$) are not trained at all. The last two matrices (connections from input/bias to reservoir) are configured in Section IV. The learning of the RC network is a fast process without local minima. 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).

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

$$\mathrm{NMSE} = \frac{\langle (y_d - y)^2 \rangle}{\sigma_{y_d}^2} \quad (7)$$

where the numerator is the mean square error of the output $y$ and the denominator is the variance of desired output $y_d$.

## III. Robot Model

We use a robot model that is part of the 2D SINAR simulator [15] in the following experiments. Its simulation environment generates the data necessary for training the RC networks. The environment of the robot is composed of several objects, each one of a particular color. Obstacles (repulsive objects) have the blue color whereas targets (attractive objects) have the yellow color. The robot model is shown in Fig. 2. 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) [15]. 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 $N(0, 0.01)$ ). 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) (summary in Table I).

The SINAR model (based on [15]) is an intelligent navigation system composed of hierarchical neural networks which learn by classical reinforcement learning algorithms. The system learns to seek targets and avoid obstacles as the robot interacts with the environment (by colliding against obstacles and by capturing targets in the environment). It also learns to distinguish targets and obstacles (which present distinct colors) by associating their respective colors to attraction or repulsion behaviors (see [15], [17]). From now on, the controllers obtained from this model will be called INASY (Intelligent autonomous NAvigation SYstem).

The INASY controllers will provide examples of navigation trajectories to a RC-based robot controller which will be called RECNA (REservoir Computing NAvigation system) from now on. The samples collected from INASY controllers (distance and color sensors, and actuators) are used to train the RECNA controller in a Matlab environment using the RCT Toolbox[1] [1]. The experimental setup is given in the following section.

## IV. Modeling Multiple Behaviors

In [14], an intelligent navigation system for mobile robots (the INASY) is identified by a RC network (learning by examples). The resulting RC-based controller generalizes for different environments, being able to navigate efficiently in all of them (avoiding obstacles and capturing targets).

The current work investigates how a single reservoir can learn multiple and conflicting robot behaviors simultaneously. Furthermore, we also need to have an efficient switching mechanism between the learned behaviors. In this section, we will train a RC network to reproduce the following robot behaviors: Environment Exploration (EE) and Target Seeking (TS). The EE behavior makes the robot explore the environment but ignoring the targets, while the TS behavior makes the robot seek and capture targets in the environment.

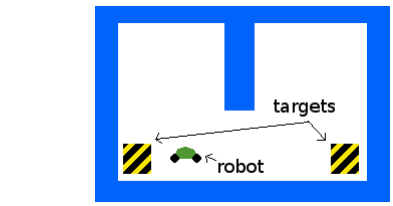
The environments used for the experiments are shown in Fig. 3. The first environment is composed of a (blue) corridor with two (yellow) targets (the targets are striped in the figure for clarification). During simulation, the robot keeps navigating through the corridor and, if desired (i.e., for the TS behavior), captures the targets (that are sequentially put back in the same locations).

The EE and TS behaviors are generated by distinct INASY controllers because they are conflicting behaviors. As the EE behavior ignores the targets in the environment, the respective INASY controller (that generates EE behavior) learned to avoid blue objects as well as yellow objects (Fig. 4(a)). On the other hand, the INASY controller that generates the TS behavior learned to avoid blue objects as well as to seek yellow objects as usual (Fig. 4(b)). See Table II for a summary. So,

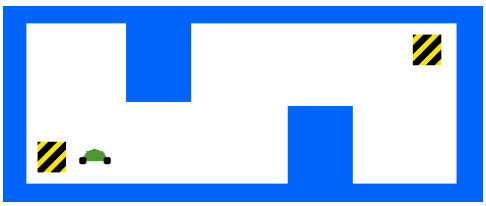
[1]This is an open-source Matlab toolbox for Reservoir Computing which is freely available at http://www.elis.ugent.be/rct

TABLE I
Robot model

| | |
|---|---|
| No. Dist. Sensors | 17 |
| No. Color Sensors | 17 |
| Range of Dist. Sens. | 300 d.u. |
| Noise on sensors | N(0,0.01) |
| Speed | 0.28 d.u./timestep |

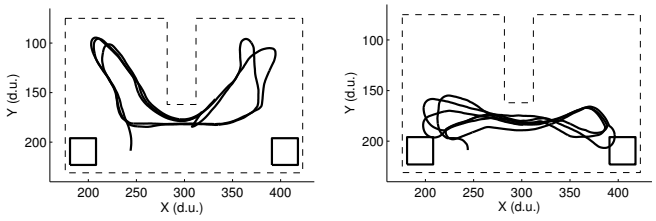



(a) E1



(b) E2

Fig. 3. Environments used for the experiments in this work. Initially, both targets are visible. After the robot captures one target, the other target is put back to its original location, making at least one target always visible. (a) Small environment with two targets and one robot. (b) Big environment with two targets and a robot.




(a) EE (b) TS

Fig. 4. Example of navigation trajectories of INASY controllers in environment E1. (a) EE exploratory behavior. (b) TS target seeking behavior.

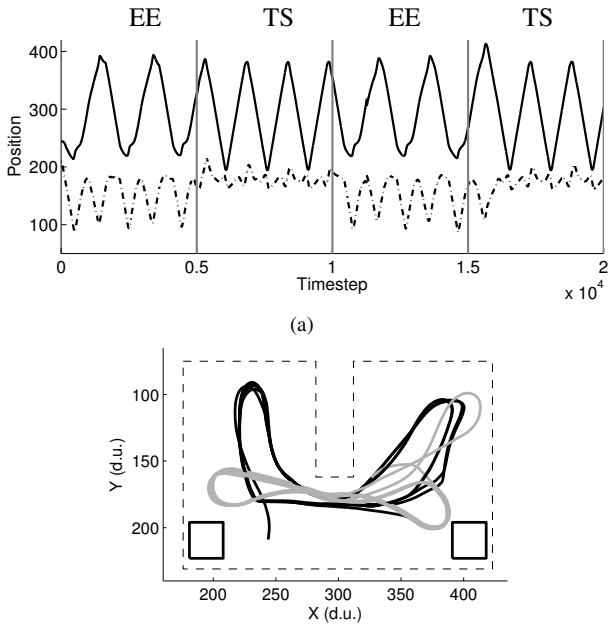
there are two teacher INASY controllers used for generating training data and one RECNA controller which will learn the conflicting navigation strategies from these INASY controllers.

In the following, we recorded the sensory and actuator samples of INASY controllers in two stages. In the first stage, the controller with EE behavior steers the robot in environment E1 (Fig. 3), exploring the environment and ignoring targets (as they were obstacles). All sensory inputs and actuators are recorded. In the second stage, the controller with TS behavior steers the robot in the same environment E1, but now generating a different trajectory towards the targets. Each stage lasted 22.500 timesteps, summing up 45.000 timesteps in total.

After getting the data which represent both behaviors individually, we will train a single RC network to reproduce these two behaviors. The RC-based navigation system (RECNA)

TABLE II
TYPE OF BEHAVIORS

| Behavior | Type | Avoids |
|---|---|---|
| EE | Exploratory | Obstacles/Targets |
| TS | Target Seeking | Obstacles |




(a)

(b)

Fig. 5. Results for environment E1. (a) The coordinates of the robot are shown for 20.000 timesteps during the testing. The solid and dashed lines are the x and y coordinates, respectively. Vertical gray lines represent the moments of behavior switching. (b) The corresponding trajectory of the robot in the Cartesian map. The solid black (gray) line represents the timesteps in which the selected behavior is the EE (TS) behavior.

learns by imitating INASY controllers [14]. However, the RECNA system will have to learn two distinct controllers (or behaviors) in one shot, and it should be able to switch between behaviors. In order to do that, an extra input is added to the RECNA controller, representing the behavior to be *selected*. If this extra input is zero (one), then the EE (TS) behavior is selected.

In the following, the parameter configuration for the RC network (of the RECNA controller) is presented. The inputs to the network are 17 distance sensors, 17 color sensors, plus 1 input for behavior selection (total of 35 inputs). The reservoir size is 600 neurons. The readout layer has 1 output unit which corresponds to the turning (direction adjustment) robot actuator (the robot has constant velocity). The connection matrix from input/bias to the reservoir ($\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$, $\mathbf{W}_{\mathrm{b}}^{\mathrm{r}}$) is initialized to -0.2, 0.2 and 0 with probabilities 0.1, 0.1 and 0.8, respectively. This parameter setting for weight matrices is not critical for the experiments.

After setting up the RC network, it was trained with the data of 45.000 timesteps (as mentioned before), of which half of the observations considered an extra input of 0 for EE behavior, and the other half considered an extra input of 1 for TS behavior. The training was done offline and in one shot using (6). After learning, the RECNA controller was evaluated in two environments. The results for environment E1 are shown in Fig. 5. The run takes 20.000 timesteps. At each 5.000 timesteps, the behavior is switched from EE to TS or vice-versa. Note that every switching implies a waiting time of 15

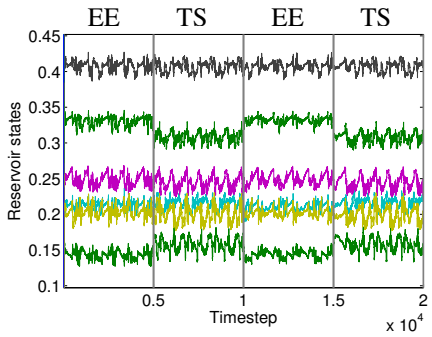



Fig. 6. Reservoir states for the RECNA controller in environment E1. The plot shows six randomly chosen states from the reservoir. Vertical lines represent the moments in which the behavior switches.

timesteps in which the robot stays still so that a short reservoir transient takes place. After this switching interval, the reservoir is ready to drive the robot according to the selected behavior. Fig. 5(a) shows the coordinates of the robot during the run (vertical lines represent the moment of switching the behavior). As we can note, the behaviors are very well defined in their respective time interval. The trajectory of the robot changes as soon as the switching occurs and a target is localized. Fig. 5(b) shows the corresponding robot trajectory for the considered robot run. The black (gray) trajectory corresponds to the time interval in which the EE (TS) behavior was selected.

The reservoir works like a rich temporal kernel which projects the input to a high dimensional dynamic space. Fig. 6 shows six randomly selected states from the reservoir when the RECNA controller was driving the robot in environment E1. It is possible to observe that the dynamics of the reservoir changes at each moment of behavior switching (given by the vertical lines in the figure). By only changing a input from 0 to 1 or vice-versa, we were able to change the dynamics of the reservoir, and consequently the behavior of the robot.

Table III shows the results for different number of neurons ($n_r$) in the reservoir. Each line in the table presents the mean values of the: training error (NMSE, as defined in (7)), training time, number of target captures and number of collisions, for 5 robot runs each of 20.000 timesteps. Each robot run is accomplished with a different stochastically generated reservoir. The training time includes the time to generate the matrix $M$ and to calculate (6) (using an Intel Core2 Duo processor-based system). During a robot run, there are 3 switching moments like in Fig. 5. The last column of the table presents the percentage of successful runs (out of 5 for each line). We can observe that as the reservoir have more neurons, the performance of the resulting RECNA controller increases (i.e., by decreasing the number of collisions), although the training time also increases. For reservoirs containing more than 400 neurons, we always get a stable RC-based controller which can perform the selected task (EE or TS) reliably. Although smaller reservoirs are not always stable for this task, it is possible to search for the best generated reservoir and use it on the navigation task.

To test the generalization capabilities of the RECNA controller, we consider a new environment (E2), different from the training environment (E1). Environment E2 (Fig. 3) is bigger than E1, and has two targets, one located in the lower-left of the environment and another in the upper-right of the environment. The results are shown in Fig. 7. We can note that the RECNA controller generalizes very well, by exploring the environment when EE behavior is turned on and capturing targets when the behavior switches to TS.

TABLE III
RESULTS FOR DIFFERENT SIZE OF RESERVOIRS - ENVIRONMENT E1

| No. Neurons ($n_r$) | Training NMSE | Training Time (s) | No. Target Captures | No. Collisions | Correct behavior |
|---|---|---|---|---|---|
| 100 | 0.88 | 5 | 12 | 20.6 | 40 % |
| 200 | 0.85 | 9 | 12.2 | 11 | 80 % |
| 400 | 0.82 | 25 | 11.8 | 0.8 | 100 % |
| 600 | 0.80 | 60 | 12.6 | 0.6 | 100 % |

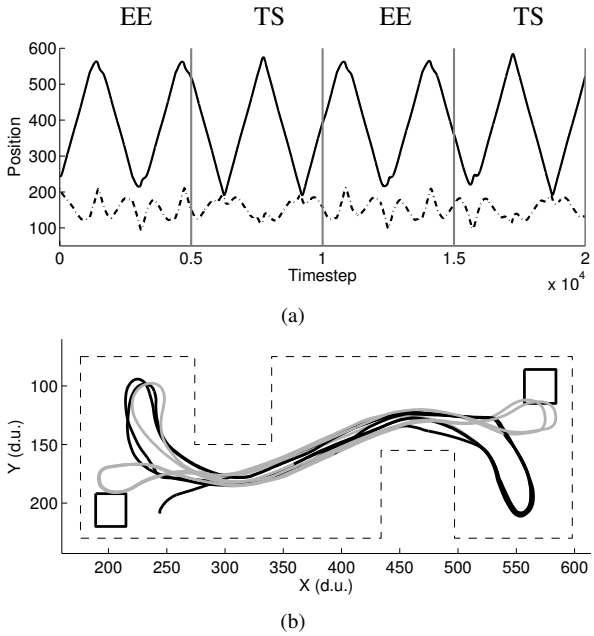
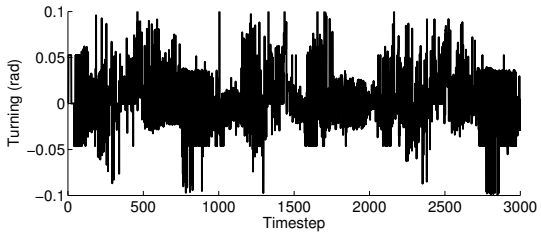


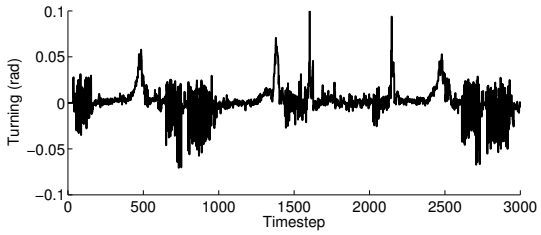

Fig. 7. Results for generalization capabilities in environment E2. (a) The coordinates of the robot are shown for 20.000 timesteps during the testing. The solid and dashed lines are the x and y coordinates, respectively. Vertical gray lines represent the moment of behavior switching. (b) The corresponding trajectory of the robot in the Cartesian map. The solid black (gray) line represents the timesteps in which the selected behavior is the EE (TS) behavior.

In this work, we have shown that we can easily imitate behaviors with RC networks. We record examples of navigation trajectories with the robot sensors and actuators, and train a Reservoir Computing network on this dataset in a supervised way. After this, the RC network performs very similarly to the original controller in the same environment and also in new environments. Fig. 8 shows the output of the original controller and of the RC-based controller. We can observe that the output of the RC network is much less noisy than the output of the teacher (INASY) controller [14].

Feedforward networks like the Multi-Layer Perceptron (MLP) are not well suited for this task of modeling multiple




(a) INASY




(b) RECNA (RC-based)

Fig. 8. Outputs (turning actuators) of controllers for EE behavior during 3.000 timesteps. (a) Output from INASY controller. (b) Output from RECNA (the RC-based) controller.

behaviors with behavior switching mechanisms. We have tried to use a MLP and backpropagation learning algorithm to reproduce the same behaviors (EE and TS) as we did with the RC network (also trying different number of hidden layers), but the MLP failed to drive the robot stably and safely (it made the robot bump to the walls repeatedly).

## V. Conclusion

This work employs Reservoir Computing (RC) as an alternative method for modeling multiple (conflicting) behaviors for mobile robots. The method corresponds to a black-box machine which learns by examples given by an intelligent navigation system in the literature [15], characterizing an imitation learning process.

In reservoir computing, the network architecture is composed of a fixed recurrent neural network (the reservoir) and a trainable readout output layer. The learning is implemented by a linear regression algorithm which guarantees convergence of the training process in a short time period and without local minima. We use a single RC network for modeling conflicting robot behaviors and we implement behavior switching by just stimulating the reservoir with an extra input which indicates which behavior should be selected. The switching mechanism is efficient, stable and robust, and works by *changing* the dynamics of the reservoir with the extra input.

This work shows just a sketch of what is possible with reservoir computing. Future work includes the study of how many behaviors can be implemented with a single reservoir. Online and autonomous learning of new behaviors by a RC network would be very desirable in the perspective of autonomous systems. In this way, reinforcement learning and reservoir computing may interplay for achieving a new and powerful class of learning systems.

In the perspective of service robotics, imitation learning can be implemented using reservoir computing techniques, making possible to teach a robot how to behave or accomplish tasks in its environment by only showing examples to the robot (moving its physical body in the environment and recording its sensors and encoders). Many applications in the real-world can easily be drawn with such an approach while a more natural interface between humans and robots is made possible.

## Acknowledgment

This research is partially funded by FWO Flanders project G.0317.05. Eric A. Antonelo is sponsored by the Special Research Fund of Universiteit Gent (BOF).

## References

[1] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "A unifying comparison of reservoir computing methods," *Neural Networks*, vol. 20, pp. 391–403, 2007.

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

[3] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[4] 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.

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

[6] 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.

[7] H. Jaeger and H. Haas, "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication," *Science*, vol. 308, pp. 78–80, April 2 2004.

[8] E. A. Antonelo, B. Schrauwen, and D. Stroobandt, "Event detection and localization for small mobile robots using reservoir computing," *Neural Networks*, vol. 21, pp. 862–871, 2008.

[9] e-puck, "http://www.e-puck.org/," 2007, e-puck education robot.

[10] E. A. Antonelo, B. Schrauwen, and J. V. Campenhout, "Generative modeling of autonomous robots and their environments using reservoir computing," *Neural Processing Letters*, vol. 26, no. 3, pp. 233–249, 2007.

[11] E. A. Antonelo, B. Schrauwen, and D. Stroobandt, "Mobile robot control in the road sign problem using reservoir computing networks," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2008.

[12] J. Tani, "Learning to generate articulated behavior through the bottom-up and the top-down interaction processes," *Neural Networks*, vol. 16, pp. 11–23, January 2003.

[13] D. Rumelhart, G. Hinton, and R. Williams, *Learning internal representations by error propagation*. MIT Press, Cambridge, MA, 1986.

[14] E. A. Antonelo, B. Schrauwen, and D. Stroobandt, "Identification of an intelligent navigation system for mobile robots using reservoir computing," in *10th Brazilian Symp. on Neural Networks (SBRN)*, 2008, (in press).

[15] 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 the International Joint Conference on Neural Networks (IJCNN)*, Vancouver, 2006, pp. 498– 505.

[16] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach," German National Research Center for Information Technology, Tech. Rep. GMD Report 159, 2002.

[17] E. A. Antonelo, M. Figueiredo, A.-J. Baerlvedt, and R. Calvo, "Intelligent autonomous navigation for mobile robots: spatial concept acquisition and object discrimination," in *Proceedings of the 6th IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Helsinki, 2005, pp. 553–557.