Modular Reservoir Computing Networks for Imitation Learning of Multiple Robot Behaviors

Tim Waegeman, Eric Antonelo, Francis wyffels and Benjamin Schrauwen*

Abstract— **Autonomous mobile robots must accomplish tasks in unknown and noisy environments. In this context, learning robot behaviors in an imitation based approach would be desirable in the perspective of service robotics as well as of learning robots. In this work, we use Reservoir Computing (RC) for learning robot behaviors by demonstration. In RC, a randomly generated recurrent neural network, the reservoir, projects the input to a dynamic temporal space. The reservoir states are mapped into a readout output layer which is the solely part being trained using standard linear regression. In this paper, we use a two layered modular structure, where the first layer comprises two RC networks, each one for learning primitive behaviors, namely, obstacle avoidance and target seeking. The second layer is composed of one RC network for behavior combination and coordination. The hierarchical RC network learns by examples given by simple controllers which implement the primitive behaviors. We use a simulation model of the e-puck robot which has distance sensors and a camera that serves as input for our system. The experiments show that, after training, the robot learns to coordinate the Goal Seeking (GS) and the Object Avoidance (OA) behaviors in unknown environments, being able to capture targets and navigate efficiently.**

I. INTRODUCTION

Autonomous mobile robots should be able to avoid obstacles while performing tasks such as target seeking, battery recharging, etc. These robots must achieve their goals in an unknown and unpredictable environment, representing constraints that are hard to fullfill in a traditional way (by modeling and manually programming all the possible events). In this context, the learning capability for a robot can be very relevant for accomplishing desired tasks.

Learning by demonstration can be very interesting if implemented correctly. The idea is that the robot can learn its task just by showing some examples of a behavior or ways of accomplishing a task. These examples are generated through a human tutor or a teacher controller. Feature extraction and high dimensionality are an important issue here. Extracting the correct features from a camera for modeling actions or behaviors is not a simple task. In this work, instead of using feature extraction methods, we use a learning architecture which can be used with the raw input data. In particular, we use Recurrent Neural Networks (RNNs) that can be trained efficiently under the paradigm of Reservoir Computing (RC) [16].

RNNs are usually very difficult to train with techniques such as Backpropagation-through-time (BPTT). The recently introduced technique for efficient and fast learning of RNNs, Reservoir Computing [16], was first introduced under the term of Echo State Networks (ESN) [9]. In RC, the output is generated by an instantaneous and linear memoryless mapping of a large untrained dynamical system (the reservoir) which is excited with one or more input signals. The reservoir is a randomly connected recurrent neural network whose connection weights are scaled so that it operates at the edge of stability, thus having a fading memory [10].

RC has proven its qualities in a broad range of applications such as speech recognition [14], time series generation [11] and in a wide range of robot applications. Complex event detection and robot localization for small mobile robots are tackled in [3] by only using few short-range distance sensors. In [2], Reservoir Computing is used for several robotics tasks such as prediction of robot position, learning of maps for unstructured environments as well as generation of robot paths in these environments. Robotic tasks which require memory on previous stimuli are learned with Reservoir Computing in [4], providing an imitation learning approach for learning delayed-response tasks like the T-maze task [4]. In [8] ESN are used to learn robot navigation behavior where the performance is comparable and even better than other algorithms from the literature.

In [5], imitation learning with RC is used for learning conflicting navigation behaviors (target seeking and exploration behavior) for mobile robots by employing a single RC Network. The RC network is stimulated by distance sensors and 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. The current work is an extension of [5] using a hierarchy of RC Networks. Our architecture is a two-layered system in which primitive behaviors are learned in the first layer. The coordination and combination is modeled in the second layer and does not depend on an extra input as in [5], but is based on visual sensor information (which can detect events from the environment).

In comparison to [5], the innovation of this approach is four-fold: we use a Webots simulation model of the e-puck robot which uses only 8 distance sensors; our model integrates infra-red sensors with camera input (multi-modal achitecture); it doesn't model explicitly switching through an extra input but uses the available sensor information to detect events; complex behavior emerges from combination of primitive modular behaviors.

This modular approach can, in principle, be used for learning multiple behaviors. As the memory of a single reservoir

*Department of Electronics and Information Systems
Ghent University, Ghent Belgium
Tim.Waegeman@UGent.be, Eric.Antonelo@UGent.be,
Francis.wyffels@UGent.be and Benjamin.Schrauwen@UGent.be

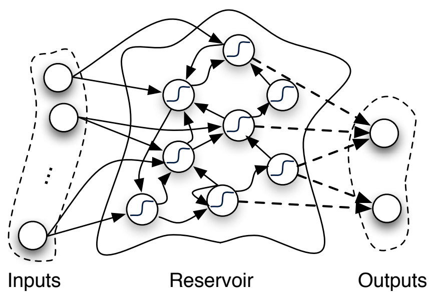

Fig. 1. Description of a Reservoir Computing network. Dashed arrows are the connections which can be trained. Solid arrows are fixed.

is limited, by using a modular RC network architecture, it can learn a higher number of robot behaviors. In addition, this architecture can handle more complex behaviors which can emerge from the imitation learning process.

Before we start describing our system and experiments we will start by a short introduction of Reservoir Computing which is the core of our technique for modeling multiple robot behaviors by imitation learning. After this, we will give a brief description of our experimental setup and the used robot model. Next, we will describe a technique to learn primitive robot behaviors such as Obstacle Avoidance (OA) and Goal Seeking (GS) by imitation. We also show that the system generalizes the learned primitive robot behaviors in unknown environments. After that, we describe how the primitive behaviors can be combined in a modular architecture with two layers. Finally we will compare the resulting behavior with a known method for combining different behaviors.

II. RESERVOIR COMPUTING

The RC network model used in this paper follows the Echo State Network (ESN) approach [9]. An ESN is composed of a discrete-time recurrent neural network (i.e., the reservoir) and a linear readout output layer which maps the reservoir states to the desired output. A schematic overview of this is given in Fig. 1. The neuron states and the readout output are updated as follows:

$$\mathbf{x}[k+1] = \tanh(\mathbf{W}_\mathrm{r}^\mathrm{r}\mathbf{x}[k] + \mathbf{W}_\mathrm{i}^\mathrm{r}\mathbf{u}[k] + \mathbf{W}_\mathrm{b}^\mathrm{r}) \quad (1)$$

$$\mathbf{y}[k+1] = \mathbf{W}_\mathrm{r}^\mathrm{o}\mathbf{x}[k+1] + \mathbf{W}_\mathrm{b}^\mathrm{o}, \quad (2)$$

where $\mathbf{u}[k]$ denotes the input at time k, $\mathbf{x}[k]$ represents the reservoir state and $\mathbf{y}[t]$ is the output. The weight matrices $\mathbf{W}_\cdot^\cdot$ 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, while all connections to the output (denoted as $\mathbf{W}_\cdot^\mathrm{o}$) are trained using standard linear regression techniques. For construction of the weight matrix $\mathbf{W}_\mathrm{r}^\mathrm{r}$, weights are drawn independently from a normal distribution with zero mean and unit variance. After construction, the matrix $\mathbf{W}_\mathrm{r}^\mathrm{r}$ is rescaled such that the largest absolute eigenvalue (spectral radius) is smaller than one, so that the system is at the edge of stability and has a fading memory [9]. In this work we always use a spectral radius equal to 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).

For many applications, the dynamics of the reservoir need to be slowed down to match the intrinsic timescale of the input data. The system's dynamics can effectively be tuned by using leaky integrator neurons [9] as follows:

$$\begin{aligned}\mathbf{x}[k+1] =& \mathbf{I}(1-\lambda)\mathbf{x}[k] + \\ & \lambda \tanh(\mathbf{W}_\mathrm{r}^\mathrm{r}\mathbf{x}[k] + \mathbf{W}_\mathrm{i}^\mathrm{r}\mathbf{u}[k] + \mathbf{W}_\mathrm{b}^\mathrm{r})\end{aligned} \quad (3)$$

where equation (1) is slightly changed by adding an additional term which takes a fraction of the the previous state $\mathbf{x}[k]$ into account. This term λ is called the leak rate. Further investigation about timescales in reservoirs and leaky integrator neurons can be found in [15], [12].
The imitation learning process uses training data which consists of both input (robot sensors) and output (actuators), which is gathered from teacher controllers. Training is performed by using linear regression on the reservoir states. For this, the reservoir is driven by an input sequence (the gathered robot sensors) which yields a sequence of neuron states using equation (3) and a sequence of outputs (the generated actuator signals) using equation (2). Next, the output connections $\mathbf{W}_\cdot^\mathrm{o}$ are trained such that the generated output signals correspond to the control signals for the actuators from the teacher controller. We determine the output connections by minimizing the least square error $J(\mathbf{w})$ according to the following equation:

$$J(\mathbf{w}) = (\mathbf{M}\mathbf{w} - \mathbf{T})^T(\mathbf{M}\mathbf{w} - \mathbf{T}), \quad (4)$$

where matrix $\mathbf{M}$ consists of a concatenation of all inputs $\mathbf{u}[k]$, reservoir states $\mathbf{x}[k]$ and outputs $\mathbf{y}[k-1]$. Matrix $\mathbf{T}$ consists of all desired outputs. By minimizing $J(\mathbf{w})$ we can determine the output connections $\mathbf{w}$.

III. ROBOT MODEL AND DATASET GENERATION

We use the Webots model of the e-puck[1] robot for our experiments. This model is extended with 8 new sensors which can measure over a larger distance (between 4 and 80 cm) than the original light intensity based distance sensors (Fig. 3). These sensors are positioned around the e-puck robot in a octagon and are modeled as single ray sensors with no noise. The model returns values between 0 and 255 (D) for the distance sensors, which we normalize:

$$D_{norm} = \frac{D - \mu_d}{\sigma_d}, \quad (5)$$

where μ_d, is the average for each distance sensor, over all samples of the known dataset. The denominator in equation (5) is the standard deviation σ_d. The e-puck is equipped with a camera which has a horizontal angle of view of 99.5 degrees. We preprocess the camera values to reduce the amount of information we feed to the reservoir. First

[1]http://www.e-puck.org

we reduce the amount of pixels to a rectangular area of 4×120 as shown in Fig. 2. Then we divide the horizontal area in 8 blocks, each of 4×15 pixels. We further reduce the image to an image of 1×8 by taking the average RGB-values over each block. If the averaged RGB-values are above a certain threshold for a red colored object we give this block the value one, and zero otherwise. Basically we converted our camera to a color sensor which is sensitive for red objects. For our experiments we use the Webots simulation environment in combination with a Reservoir Computing MatLab toolbox created at our research group. This toolbox[2] gives the experimenter the ability to quickly set up experiments. To reduce the time of every experiment we run the simulation at 5 to 15 times faster than real time, depending on the performance of the system used for the experiments. Because of our configuration concerning the camera image, we do not have a fixed synchronization between Webots and Matlab. This means that the amount of samples per second depends on the processor load of the system, although the effect on the resulting behaviors is unnoticeable.

IV. LEARNING PRIMITIVE BEHAVIORS

In this section, we will elaborate on the training of RC networks for two primitive behaviors. The first behavior is Obstacle Avoidance (OA) and uses only the distance sensors as input, whereas the second behavior is "Goal Seeking" (GS) which drives the robot towards the target. The last behavior uses only camera inputs. We will describe the controller of each primitive behavior and give more details about the RC networks used for learning. Finally we will show the generalization capabilities of each behavior, by putting the e-puck robot in a different environment during testing, depicted in Fig. 3 (right).

A. Obstacle Avoidance behavior

We use a controller to generate the OA behavior which uses a variant of the Braitenberg algorithm [6]. In Figure 4(a)

[2]RC toolbox: http://reslab.elis.ugent.be/RCtoolbox

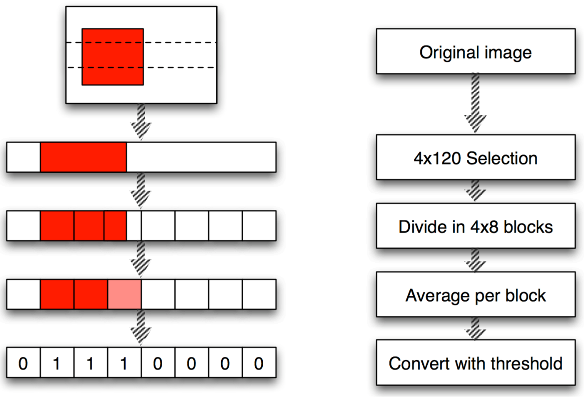

Fig. 2. The different steps of the camera image preprocessing is shown on the right together with an example of every step on the left.

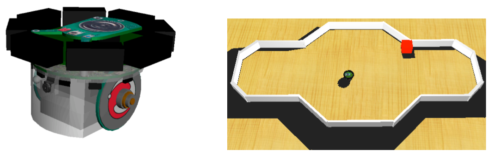

Fig. 3. Robot model (left) and simulation environment (right) with a red object (target) at the upper-right corner and the robot in the middle.

we give an example of a trajectory generated by the controller in the training environment used for generating the dataset. Similar work in [7] uses a Liquid State Machines [13] (counterpart of Echo State Networks for Spiking Neural Networks) to imitate Braitenberg vehicles. The Braitenberg controller generates examples of trajectories which are used to train the OA RC network.

To learn this behavior we use a reservoir with 300 neurons, although less is also possible because of the linear characteristics of the Braitenberg algorithm. As we mentioned shortly, we use the 8 distance sensors as input signals, which are normalized between -1 and 1. The readout layer has 2 output units which corresponds to the left and right wheel speed. The connection matrix from input to the reservoir ($\mathbf{W}_i^r$) has elements which are 0 with a probability of 0.4. The other elements are -0.1 or 0.1 with equal probability. The reservoir has a spectral radius of 0.9, a connection fraction of 0.9 and a leak rate of 0.5.

During training of the OA behavior, we insert a cube in the training environment and we move it around at times chosen by the experimenter as illustrated in Fig. 4(a). The motivation for this is that the system should have rich enough information to learn a behavior that is able to generalize.

After training, the system (RC network) produces the same behavior as the controller and is able to generalize as shown in Fig. 5(a). This means that the e-puck acts exactly the same way as the controller even in a different environment, independent of the initial position. In Fig. 5(a) we recorded the position of the robot while testing the generalization capabilities during 4000 samples.

B. Goal Seeking behavior

The GS controller generates behavior that makes the robot rotate when no red object is in the field of view of the camera.

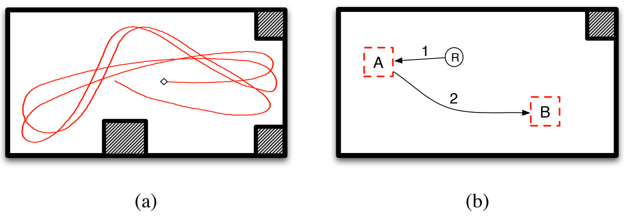

Fig. 4. Training environments. (a) The training environment for the OA RC network. During training we move one extra obstacle to random positions at random moments. This is done to prevent the controller to generate the same trajectory. (b) The training environment for the GS RC network where the red object *A* (dashed square) is moved to *B* when the robot *R* approaches the object within a specified distance.

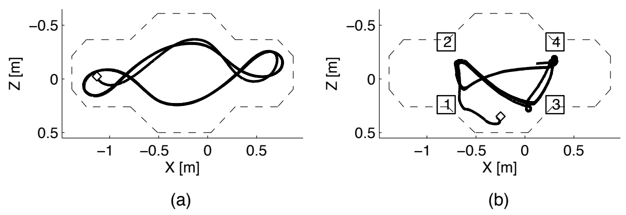

Fig. 5. (a) Robot trajectory generated by the OA system in a test environment (different then during training). (b) Robot trajectory of the GS behavior. The dashed line is the border of the test environment while the start position is marked in both plots with a small square. Labeled squares represent the sequence of the goal appearances.

When a red object becomes visible, the robot will drive to the object while trying to centralize the object in its camera view. At a certain distance from the red object, the robot will stop moving.

For this behavior we use a reservoir with 800 neurons. The input for this reservoir contains 8 values which represent the camera image after preprocessing and normalization. This normalization is achieved by using an equation similar to equation (5). The readout layer has also 2 output units which corresponds to the left and right wheel speed. The connection matrix from input to the reservoir ($\mathbf{W}_\mathrm{i}^\mathrm{r}$) has elements which are 0 with a probability of 0.5. The other elements are -0.09 or 0.09 with equal probability. The reservoir has a spectral radius of 0.9, a connection fraction of 0.8 and a leak rate equal to 0.9.

During training of the GS behavior, we insert a red cube in a different environment than the training environment of the OA behavior and move it around to a different position if the robot is close to the red cube. This is done by a supervisor controller who simulates a catch when the robot approaches the object. This training process is illustrated in Fig. 4(b).

After training, the system (RC network) produces the GS behavior and is able to generalize as shown in Fig. 5(b). To show this we put a red object at position 1. When the e-puck reaches the object, we move the object to position 2, 3 and 4. We repeat this process during 4000 samples and observe that the system is able to produce a behavior that is consistent.

V. Learning Multiple Behaviors

In this section we will describe how we create a switching/combining mechanism to establish complex behavior by switching between primitive behaviors based on events.

The idea behind the hierarchical modular RC network is to be able to get a complex control system without having to write a controller that combines the primitive behaviors. We try this by using the available information from each primitive behavior.

We use a hierarchy of modular RC networks where we have the primitive behaviors in a first layer and the switching mechanism in a second layer. The inputs to the primitive behaviors in the first layer together with there normalized outputs, is fed to the second layer as input for our switching mechanism. The output of each primitive behavior is normalized by dividing the output with the maximum output value possible (300). The inputs for OA and GS are the

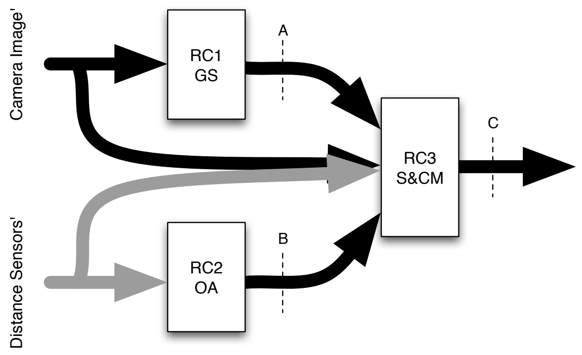

Fig. 6. The modular topology with the two RC networks of the primitive behaviors, in the first layer, and the switch and combination mechanism (S&CM), in the second layer. The GS system has the normalized preprocessed camera image as input and the wheel speeds as output. The OA system on the other hand has normalized distance sensors as input and also wheel speeds as output. The third RC network (S&CM) has both the normalized preprocessed camera image and distance sensors as input together with both normalized output wheel speeds of the OA and GS RC network, while the output of the S&CM system are normalized wheel speeds. In *A* and *B* the normalization is done by dividing the values through the maximum speed value possible. In *C* the normalized output is converted back by multiplying it with the same maximum speed value possible.

normalized distance sensors and preprocessed camera image, respectively. The output of the third system (S&CM) in the second layer, is the normalized wheel speed that is fed back to the e-puck robot after multiplying it with again the maximum output value possible. The topology of our system is shown in Fig. 6.

For this mechanism we use a reservoir with 1200 neurons. The reservoir has 20 inputs. Eight of them represent the preprocessed camera image, eight of them represent the normalized distance sensors and four inputs are respectively the normalized outputs units of the OA and the GS reservoirs. The readout layer also has 2 output units which correspond to the left and right wheel speed. The connection matrix from input to the reservoir ($\mathbf{W}_\mathrm{i}^\mathrm{r}$) has elements which are -0.1 and 0.1 with equal probability. The bias to reservoir ($\mathbf{W}_\mathrm{b}^\mathrm{r}$) is set

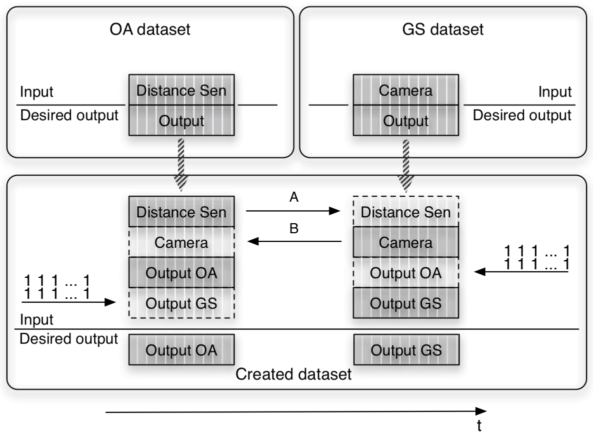

Fig. 7. Concatenation of OA and GS datasets. Dashed rectangles represent the data that is not available from the original dataset and has to be filled with other data. Arrows A and B, represent the origin of this data. The time (t) is represented by the horizontal arrow that is orientated to the right.

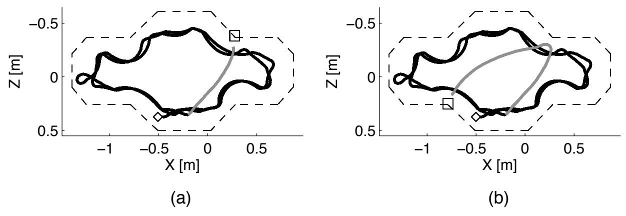

Fig. 8. (a) The switch to a RP (Red Period). (b) After changing the position of the red object the robot finds the new position and drives to it. The black and gray lines represent the OA and GS behavior, respectively. The start position is marked in both plots with a square.

as 0.05 with a connection fraction of 1. The reservoir has the same spectral radius and connection fraction as GS and uses a leak rate of 0.9.

We train the reservoir of this mechanism by constructing a new dataset with data from the primitive behaviors. We do this by concatenating the individual datasets as shown in Fig. 7. The input layer of this reservoir has 20 values, as mentioned above. If we just concatenate the individual datasets we would be missing some unknown camera and distance values in respectively the OA and GS datasets which are represented by the dashed rectangles. Each block represents an $m \times n$ matrix where n is the amount of samples in the time direction (arrow t) and m represents the amount of values which depends on the type of block (m=8 for the camera and distance sensors and m=2 for the outputs). Because we trained both behaviors in different environments we can use the GS camera values to fill the gap in the OA part (arrow B) and the distance values of OA to fill the gap in the GS part (arrow A). Each part consist now of 8 distance values, 8 camera values and 2 output speed values. We fill the two missing speed values in the OA part with static input; both equal to one which is arbitrary chosen and do the same for the GS part. This is indicated by an arrow with ones for the left and right wheel speed. By constructing the datasets as described in Fig. 7 we have 20 values for the input layer in both parts. The output layer is trained using teacher forcing, by forcing the OA speeds during the training of the OA part and the GS speeds during the training of the GS part of the constructed dataset as illustrated in Fig. 7.

One can ask why we did not record the distance sensors and camera values while recording the datasets of each separate behavior, preventing us of copying parts of each dataset to the other (arrow A and B in Fig. 7)? We have done this intentionally because in this way there is only a relationship between the distance sensors (camera values) and the desired output in the OA (GS) part, respectively.

As shown in Fig. 8 this RC network produces a behavior that is rather combining behaviors than switching between them.

When there is no red object inside the environment the system produces a combination of the OA and the turning of the GS behavior. Therefore the resulting behavior has the characteristics of a wall-follower when the initial position of the e-puck is close to a border of the environment. After 1164 samples we put a red object in the environment (RP: Red Period) and change the position of the object when the e-puck reaches it. The e-puck reacts when it detects the red object in its camera image and drives straight to it. When we remove the red object again from the environment after 716 samples, the e-puck returns to the previous behavior. In Fig. 9, one can observe a change in the reservoir states which causes a change in the final wheel speeds and results in a different trajectory, according to the behavior. We notice the moments where the robot switches back and forth between the two behaviors.

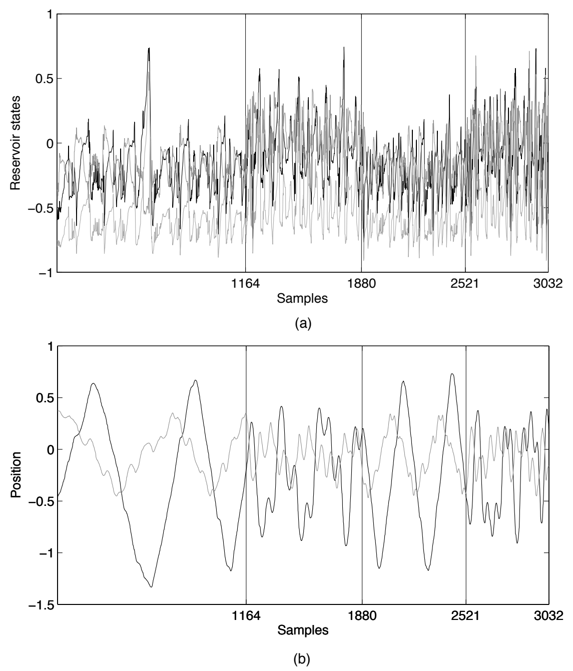

Fig. 9. Results of modeling multiple behaviors. (a) Reservoir states of the switch and combination system (S&CM). The plot shows 3 randomly chosen states from the reservoir. Vertical lines represent the switching moments of the behaviors. (b) The coordinates of the robot are shown during 4 switching moments. The black and gray lines are the x and z coordinates respectively. Vertical lines represent the switching moments.

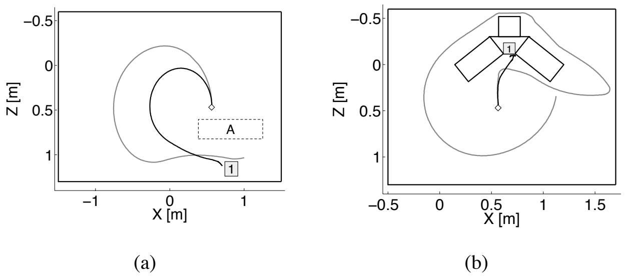

Fig. 10. (a) Comparison of generated trajectories from the RC method (gray) and vector method (black) with a red object at position 1 in a rectangular environment. In scenario 2 of the last experiment there is an obstacle A used. (b) Comparison of robot trajectories from RC method (gray) and vector method (black) in a rectangular environment with a red object (1) in the corner of an obstacle. The start position is marked in both plots with a small square.

We compare the achieved behavior with the existing vector method [1] for combining different behaviors. In this method each influence of a behavior is seen as a vector which points in the direction of the movement. The resulting movement

TABLE I
MEASUREMENTS OF BOTH METHODS IN 2 DIFFERENT SCENARIOS

	RC method		Vector method	
	time	distance (m)	time	distance (m)
Scenario 1	4m 11s	7.28	2m 3s	3.18
Scenario 2	4m 47s	7.82	9m 38s	11.04

is that of the combination of each vector which we calculate by averaging the left and right wheel speeds of both the OA and GS behavior. If we position the e-puck robot in a rectangular environment with his camera directed to the top border of the environment and a red object at position 1, we notice a difference in movement for both methods as shown in Fig. 10(a) without the obstacle *A*.

We compare our technique to the vector method in the risky configuration shown in Fig. 10(b) (where a red object is situated in a corner among obstacles). With the vector method, both vectors (for obstacle avoidance and goal seeking) are directed in opposite directions which causes the robot to stop. This figure also shows that the hierarchical RC network can drive the robot correctly in this environment, by switching the behavior to obstacle avoidance and deviating from the obstacles.

In a different experiment we compare both techniques during 2 scenarios. In scenario 1 there is no obstacle between the robot and the red object (Fig. 10(a) without obstacle *A*) while scenario 2 is with an obstacle in between (robot can not see the red object, shown in Fig. 10(a) with obstacle *A*). We repeat this experiment for each method 5 times per scenario and measure the time and distance that the robot takes to reach the red object. In Table I we notice after averaging over 5 runs that the RC method is consistent in both scenarios while the vector method scores very well in scenario 1 but less in scenario 2.

VI. CONCLUSION

In this work we described a technique based on Reservoir Computing (RC) for the modeling of multiple (conflicting) behaviors for autonomous mobile robots. We use an obstacle avoidance behavior, and a behavior that searches and approaches a red object. Behaviors can be learned in separate modules by giving a module examples of the desired behavior. By testing the learned behaviors of the robot in an environment different from the training environment we showed the generalization capabilities of our system.

After learning primitive behaviors we extended our system to a two-layered structure where each primitive behavior module is located in the first layer. In the second layer, a module was trained to switch between the modules of the first layer, based on all sensor inputs and control signals. This enables the robot to search for a target (red object in this paper) without hitting any obstacles while doing so.

We compared this RC method for combining two behaviors with the existing vector method [1] which has a similar movement as result in some situations. The experimental results show that under particular environments, the RC method performs better thant the vector method (e.g. our method can correctly perform obstacle avoidance for risky collision situations).

In future work we plan to investigate the applicability of our system for more than two behaviors as well as to a real e-puck robot. It would also be desirable that new behaviors can be learned during operation of the robot by only showing examples (e.g. moving its physical body in the environment and recording its sensors and encoders).

VII. ACKNOWLEDGMENTS

This research is partially funded by the FWO Flanders project G.0088.09, the FP7 funded ORGANIC project (FP7-231267), and the Photonics@be Interuniversity Attraction Poles program (IAP 6/10), initiated by the Belgian State, Prime Minister's Services, Science Policy Office.

REFERENCES

[1] Alex M. Andrew. Behavior-based robotics by ronald c. arkin, mit press, cambridge, mass., 1998, xiv+491 pp, isbn 0-262-01165-4. *Robotica*, 17(2):229–235, 1999.

[2] Eric A. Antonelo, Benjamin Schrauwen, and Jan Van Campenhout. Generative modeling of autonomous robots and their environments using reservoir computing. *Neural Processing Letters*, 26(3):233–249, 2007.

[3] Eric A. Antonelo, Benjamin Schrauwen, and Dirk Stroobandt. Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, 21:862–871, 2008.

[4] Eric A. Antonelo, Benjamin Schrauwen, and Dirk Stroobandt. Mobile robot control in the road sign problem using reservoir computing networks. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2008. (accepted).

[5] Eric A. Antonelo, Benjamin Schrauwen, and Dirk Stroobandt. Modeling multiple autonomous robot behaviors and behavior switching with a single reservoir computing network. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 1843–1848, Singapore, 10 2008.

[6] Valentino Braitenberg. Vehicles: Experiments in synthetic psychology. *MIT Press, Cambridge, Massachusetts*, 1984.

[7] Harald Burgsteiner. Imitation learning with spiking neural networks and real-world devices. *Eng. Appl. Artif. Intell.*, 19(7):741–752, 2006.

[8] Cédric Hartland and Nicolas Bredeche. Using Echo State Networks for Robot Navigation Behavior Acquisition. In *ROBIO 07*, pages 201–206, Sanya China, 2007.

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

[10] Herbert Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, 2001.

[11] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 308:78–80, April 2 2004.

[12] Herbert Jaeger, Mantas Lukosevicius, and Dan Popovici. Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20:335–352, 2007.

[13] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[14] Benjamin Schrauwen, Michiel D'Haene, David Verstraeten, and Jan Van Campenhout. Compact hardware liquid state machines on fpga for real-time speech recognition. *Neural Networks*, 21:511–523, 2008.

[15] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, 2007.

[16] David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20:391–403, 2007.