Supervised Learning of Internal Models for Autonomous Goal-Oriented Robot Navigation using Reservoir Computing

Eric A. Antonelo and Benjamin Schrauwen

Abstract— In this work we propose a hierarchical architecture which constructs internal models of a robot environment for goal-oriented navigation by an imitation learning process. The proposed architecture is based on the Reservoir Computing paradigm for training Recurrent Neural Networks (RNN). It is composed of two randomly generated RNNs (called reservoirs), one for modeling the localization capability and one for learning the navigation skill. The localization module is trained to detect the current and previously visited robot rooms based only on 8 noisy infra-red distance sensors. These predictions together with distance sensors and the desired goal location are used by the navigation network to actually steer the robot through the environment in a goal-oriented manner. The training of this architecture is performed in a supervised way (with examples of trajectories created by a supervisor) using linear regression on the reservoir states. So, the reservoir acts as a temporal kernel projecting the inputs to a rich feature space, whose states are linearly combined to generate the desired outputs. Experimental results on a simulated robot show that the trained system can localize itself within both simple and large unknown environments and navigate successfully to desired goals.

I. INTRODUCTION

Autonomous robots should be able to learn their abilities through interaction with the environment. Learning its own internal rules for sensory-motor coupling in close interaction with the environment represents a higher degree of autonomy for a robot. This also implies adaptation and robustness to noise and unpredictable events.

Standard models of deliberative systems for autonomous navigation rely on a predefined set of rules for path planning. A lot of design effort has to be put in creating a map of the environment and modeling all possible events and situations during robot navigation. It can also be very computationally expensive. The probabilistic SLAM approach represents the state-of-the-art in simultaneous localization and mapping [5], but it lacks in adaptation and learning capabilities and usually requires fully equipped robot platforms with expensive laser scanners for environment mapping. Furthermore, it usually has rather large computational requirements.

Recent adaptive navigation models have been proposed in the literature which either try to solve challenging real-world problems [13], [19] or are oriented towards modeling an animal's capability for spatial navigation [4], [20], [7]. Machine learning techniques are used in both contexts and more biologically-inspired methods can be preferred depending on the task and the context. This work could be situated

in between the real-world robotic problems and biologically-inspired models. In this context, we employ a biologically plausible method called Reservoir Computing (RC) [23] to train a mobile robot controller in a supervised way. RC is a unifying term for techniques which efficiently train Recurrent Neural Networks (RNNs). They are know as Echo State Network (ESN) [9] for analog neurons or Liquid State Machines (LSM) [11] for spiking neurons. In these systems, the RNN is a *reservoir* of randomly generated nodes which projects the input to a high-dimensional space (acting like a non-linear kernel). The reservoir is not trained at all, but only a readout output layer, usually by linear regression methods. Fig. 1 shows an example of a RC network.

This work proposes a hierarchical architecture composed of two reservoir modules, one for localization and another for navigation. The localization reservoir receives input from only 8 low-accuracy distance sensors and determines the current and previously visited robot room. The mapping between reservoir states to the predicted rooms is learned in a supervised way from examples of robot trajectories in the considered environment. In a second learning stage, the navigation reservoir is trained with several examples of routes from a source location to a goal location, using inputs from the localization reservoir (predicted locations), the distance sensors and the desired goal location (given as input to the system). So, this navigation module integrates different types of input and can simultaneously learn reactive (obstacle avoidance) and deliberative (sequence of decisions) behaviors (also shown in [3] for delayed response tasks like the road sign problem).

It is important to see that the internal reservoir memory is essential for learning an internal model of the environment and of the robot task. The recurrent reservoir has states which reflect the recent history of inputs, representing a short-term memory capable of combining and dealing with different sources of information. Its inherent capability for modeling temporal non-linear systems makes it very interesting for constructing internal models. As argued by J. Tani [21], behavior-based systems without internal models are blind. So, by combining reservoir computing and goal-oriented navigation, we aim at creating a unifying and efficient method for imitation learning of deliberative and reactive behaviors (and its underlying internal models).

The ability to learn an implicit map of the environment can also be found in rodents. Their hippocampus have spatial processing units, called place cells, which responds maximally for specific parts of the environment (called place fields), effectively mapping the whole environment [15]. A similar
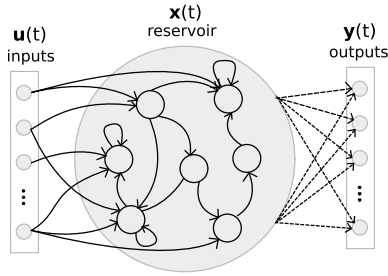
Fig. 1. Reservoir Computing network. The reservoir functions like a temporal kernel which projects the input to a rich feature space. Solid lines represent fixed connections. Dashed lines define connections which should be learned.

approach is used in this work where distinct outputs in the localization module are used to encode specific locations in the environment, given low dimensional sensory input, resembling hippocampal place cells of rodents. Although our model does not use any form of path integration (e.g., odometry), the reservoir provides a short-term memory of previous inputs, making it possible that the robot maintains an estimate of its current location for additional timesteps even in the absence of sensory input.

Relevant research in learning the localization capability for small mobile robots using RC can be found in [2] in a supervised learning approach and in [1] in an unsupervised way. In these context, the current work is the first to integrate localization and navigation using the Reservoir Computing paradigm. Several recent works have been using RC in robotics: in [16] for mobile robot modeling and control, in [10] for movement generation, in [17] for motor control, and in [22] for underwater robot control.

The advantages of the current approach are three-fold: no special environment landmarks are required; it works for small mobile robots having few low-accuracy distance sensors; and deliberative and reactive navigation components are learned in an imitation-based way with the same hierarchical architecture. We also show in this paper that robot kidnapping can easily be overcome even if the robot is not trained with this situation, showing a generalization capability of the proposed system.

The experiments are accomplished with a simulation model of the e-puck robot extended with longer-range ([5cm-80cm]) infra-red sensors in the Webots environment. It is shown that proposed system can learn with examples to drive a robot to a desired goal location in simple and bigger (9 rooms) environments using only 8 low-accuracy sensors and the goal location as input. This paper is organized as follows. In Section II, reservoir computing, the robot model and the hierarchical model are described. Experiments and its associated results are presented in Section III. Finally, conclusions and future work are given in Section IV.

II. METHODS

A. Reservoir Computing

The RC model we use is based on the Echo State Network (ESN) approach [8]. The state update equation for the

reservoir is given by:

$$\mathbf{x}(t+1) = f((1-\alpha)\mathbf{x}(t) + \alpha(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \mathbf{W}_{\text{res}}\mathbf{x}(t) + \mathbf{W}_{\text{bias}}^{\text{res}})), \quad (1)$$

where: $\mathbf{u}(t)$ represents the input at time t; $\mathbf{x}(t)$ is the reservoir state; α is the leak rate; and $f() = \tanh()$ is the hyperbolic tangent activation function; $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{bias}}^{\text{res}}$ are the weight matrices from input and bias to reservoir, respectively and $\mathbf{W}_{\text{res}}$ represents the recurrent connections between internal nodes of the reservoir. The initial state is $\mathbf{x}(0) = \mathbf{0}$. A standard reservoir equation (without the leak rate) is found when $\alpha = 1$.

The output of the RC network $\mathbf{y}(t)$ is given by a linear combination of the reservoir states plus a bias:

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

The non-trainable weights $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{res}}$ are randomly initialized. Each element of the connection matrix $\mathbf{W}_{\text{res}}$ is drawn from a normal distribution with zero mean and unit variance. This matrix is rescaled so that the reservoir has the echo state property [8], that is, its spectral radius $|\lambda_{max}|$ (the largest absolute eigenvalue) of the linearized system is smaller than one [8]. This means that the reservoir should have a fading memory such that if all inputs are zero, the reservoir states also approach zero within some time period. For most applications, the best performance is attained with a reservoir that operates at the edge of stability $|\lambda_{max}| = 0.98$. The initialization of the reservoir parameters are given in Section III.

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; n_s is the number of samples.

The training of the output layer consists of finding the weights $\mathbf{W}_{\text{out}}$ which minimizes the sum of squared errors

$$\sum_{t=1}^{n_s} (\mathbf{y}(t) - \hat{\mathbf{y}}(t))^2,$$

by the Least Squares Method:

$$\mathbf{M}\mathbf{W}_{\text{out}} = \hat{\mathbf{Y}} \quad (3)$$
$$\mathbf{W}_{\text{out}} = (\mathbf{M}^\top\mathbf{M})^{-1}\mathbf{M}^\top\hat{\mathbf{Y}}, \quad (4)$$

where: $\mathbf{M}$ is the matrix of size $n_s \times (n_r + 1)$ with the generated reservoir states collected row-wise where the last column of $\mathbf{M}$ is composed of 1's (representing a bias). The desired outputs (e.g., location or motor actuators) are collected row-wise into a matrix $\hat{\mathbf{Y}}$.

Note that the other matrices ($\mathbf{W}_{\text{res}}, \mathbf{W}_{\text{in}}, \mathbf{W}_{\text{bias}}$) are not trained at all. The last two matrices (connections from input/bias to reservoir) are configured in Section III. The learning of the RC network is a fast process without local minima, which is not the case for algorithms such as BackPropagation-Through-Time (BPTT).

The supervised learning procedure consists of two stages as follows. First, it is necessary to generate several examples of robot trajectories from a source location to a goal location (see Section II-B). All required data are recorded during

this stage such as the distance sensors and the goal location (input) and the robot location and desired motor actuators (output). The second stage involves the training of the RC networks with the recorded data (see Section II-C). Afterwards, the trained system can be used to drive the robot to specific target locations given as input.

B. Robot Model and Dataset Generation

The robot model used in the following experiments is the simulated e-puck robot [14] extended with 8 infra-red sensors which can measure distances in the range [5-80] cm. We use the Webots simulation environment [12] for data generation and navigation experiments, providing physically-realistic simulations (the simulator detects collisions and simulates physical properties of objects, such as the mass, the velocity, the inertia, the friction, the spring and damping constants, etc.). A simulated timestep in Webots takes 32 ms. The original simulation model of the e-puck has a 5.20 cm diameter (10 cm when modified with the extra turret for the infra-red sensors) and its actuators are 2 stepper motors. In the simulation, the robot wheels have a radius of 2 cm.

While the robot navigates in the Webots simulation environment, a dataset (with sensors, actuators, and locations) is recorded into a Matlab environment (communication implemented with TCP/IP sockets). The robot controller used to generate these training datasets is composed of a simple linear obstacle avoidance algorithm (the Braitenberg vehicle [6]) which is steered by a higher level planner (e.g., a program or a human supervisor). The speed (steps/second) of the robot is variable (the maximum speed is 1000 steps per second). In this work, the actuator is limited to the interval $\pm[0, 300]$ steps/s (or $\pm[0, 3.77]$ cm/s).

C. Hierarchical Architecture

The proposed architecture is based on the following principles: autonomous navigation is achieved by a process of imitation leaning which trains the proposed architecture with examples of correct goal-oriented trajectories; and goal-oriented navigation should be achieved by learning a spatial representation of the environment by the robot's own sensors (embodied cognition). The second point implies that the system does not know the map of the environment a priori. The reservoir architecture and training procedure follow the Reservoir Computing paradigm, which has been associated to cerebellar functioning in real brains [26]. We will call this architecture from now on as Reservoir Computing Hierarchical Controller (RC-HC).

The RC-HC architecture is composed of two reservoirs: the localization reservoir and the navigation reservoir (see Fig. 2). The localization module predicts the current robot location as well as the previously visited robot location given only 8 distance sensors as input. The reservoir projects the robot's sensors to a high-dimensional space whose states are linearly combined to detect the robot location [2]. This mapping is learned with linear regression (see Section II-A). This reservoir has a low leak rate α which provides more memory to hold information on past inputs. The output
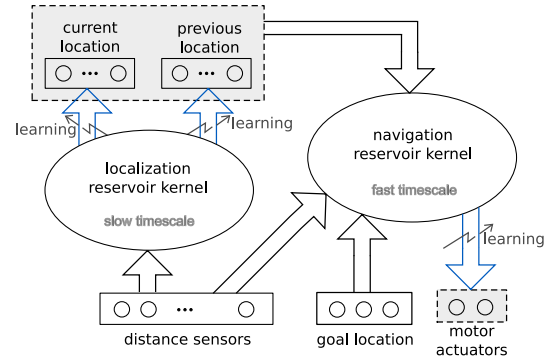

Fig. 2. Hierarchical architecture with localization and navigation modules. The navigation and localization reservoirs are randomly generated recurrent networks which are not trained, but left fixed. Trainable components are shown in shaded rectangles. The sensory input feeds both reservoirs, being mapped to a high-dimensional space, where learning occurs. The navigation reservoir receives input also from the localization module and the target location to determine the desired motor actuators.

layer of the localization module (see Fig. 2) creates a spatial representation of the environment which is comparable to the representation provided by the place cells found in the hippocampus of rats (areas CA1 and CA3, [15]). These place cells increase activity whenever the rat (robot) is in a specific region of its environment (which defines the place field of the cell).

The navigation reservoir accounts for steering the robot given several sources of information. It receives input from the robot distance sensors so that it can efficiently avoid obstacles and input from the localization module and the goal location for decision making (planning). All these inputs are integrated in one *fast reacting* reservoir (with a high leak rate) whose states are linearly combined to set the motor actuators for the left and right wheels.

The learning process is divided in two stages. First, the localization module is trained with examples of robot trajectories to detect the current and previously visited robot room using the controller described in last section. After this, we train the navigation module with new examples of robot trajectories, but now using the prediction of the trained localization module as input.

By rewriting equations (1) and (2) for the localization module, we get:

$$
\begin{aligned}
\mathbf{x}^{\text{loc}}(t+1) = \quad & f((1-\alpha_{\text{loc}})\mathbf{x}^{\text{loc}}(t) + \alpha_{\text{loc}}(\mathbf{W}_{\text{in}}^{\text{loc}}\mathbf{u}_{\text{dist}}(t)+ \\
& \mathbf{W}_{\text{res}}^{\text{loc}}\mathbf{x}(t) + \mathbf{W}_{\text{bias}}^{\text{res}})),
\end{aligned}
$$

$$
\mathbf{y}^{\text{cloc}}(t+1) = \quad g(\mathbf{W}_{\text{out}}^{\text{cloc}}\mathbf{x}^{\text{loc}}(t+1) + \mathbf{W}_{\text{bias}}^{\text{out\_cloc}}) \qquad (5)
$$

$$
\mathbf{y}^{\text{ploc}}(t+1) = \quad g(\mathbf{W}_{\text{out}}^{\text{ploc}}\mathbf{x}^{\text{loc}}(t+1) + \mathbf{W}_{\text{bias}}^{\text{out\_ploc}}), \qquad (6)
$$

where $\mathbf{y}_{\text{cloc}}$ and $\mathbf{y}_{\text{ploc}}$ are vectors of size n_l representing the predicted current and previous robot locations, respectively; n_l is the number of locations or rooms in the environment and $g(\mathbf{x})$ is a winner-take-all function which gives +1 for the highest input and -1 otherwise. The other parameters and variables have the same meaning as the ones in Section II-A, but have new subscripts for identifying the localization reservoir.

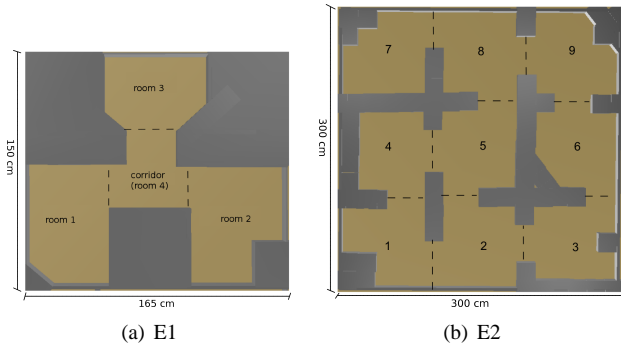Analogously, the equations for the navigation module are

Fig. 3. Webots environments used for experiments. (a) Environment (165 cm x 150 cm) with 3 goal rooms and a connecting corridor. (b) Large environment (300 cm x 300 cm) with 9 rooms (goal locations are 1, 3, 7 or 9). Dashed lines represent boundary limits between rooms.
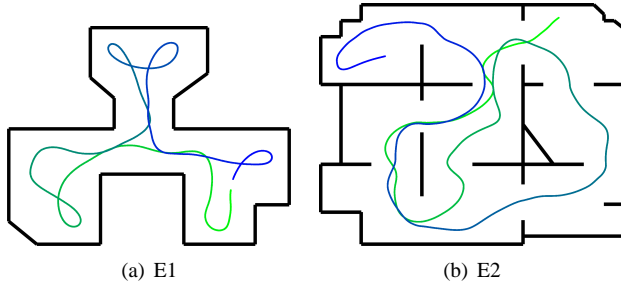

(a) E1   (b) E2

Fig. 4. Samples of robot trajectories used as training examples for the RC-HC controller. (a) Trajectory in E1. (b) Trajectory in E2.

as follows:

$$
\begin{aligned}
\mathbf{x}^{\mathrm{nav}}(t+1) =\ & f((1-\alpha_{\mathrm{nav}})\mathbf{x}^{\mathrm{nav}}(t) + \alpha_{\mathrm{nav}}(\mathbf{W}_{\mathrm{in}}^{\mathrm{nav}}\mathbf{u}_{\mathrm{multi}}(t) \\
& + \mathbf{W}_{\mathrm{res}}^{\mathrm{loc}}\mathbf{x}(t) + \mathbf{W}_{\mathrm{bias}}^{\mathrm{res}})), \\
\mathbf{y}^{\mathrm{nav}}(t+1) =\ & g(\mathbf{W}_{\mathrm{out}}^{\mathrm{nav}}\mathbf{x}^{\mathrm{nav}}(t+1) + \mathbf{W}_{\mathrm{bias}}^{\mathrm{out\_nav}}) \qquad (7)
\end{aligned}
$$

where $\mathbf{y}_{\mathrm{nav}}$ is a vector with the speeds for the left and right wheels of the robot; and $\mathbf{u}^{\mathrm{multi}}(t)$ is a concatenated input vector consisting of the distance sensors, the current and previous predicted locations, and the goal location

$$
\mathbf{u}^{\mathrm{multi}}(t) = [\mathbf{u}_{\mathrm{dist}}^{T}(t)\mathbf{y}_{\mathrm{cloc}}^{T}(t)\mathbf{y}_{\mathrm{ploc}}^{T}(t)\mathbf{u}_{\mathrm{goal}}^{T}(t)]^{T}
$$

.

The weight matrices $\mathbf{W}$ in Equations (5), (6) and (7) are trained using linear regression as explained in Section II-A. All other weight matrices (connecting to the reservoir) are randomly generated at the beginning of the experiment.

III. EXPERIMENTAL RESULTS

We have evaluated the proposed RC-HC hierarchical architecture in two environments. Environment E1 is composed of three rooms connected by a central corridor (see Fig. 3). A second, larger environment E2 is made of 9 rooms with open doors connecting them. For the first environment, there are two training datasets, one consisting of 500.000 samples (4 hours and a half of simulation time) for training the localization module in a first step and the other one consisting of 100.000 samples for training the navigation reservoir in a second step. These training datasets contain examples of trajectories of a robot continuously going from

TABLE I

PARAMETER CONFIGURATION FOR EXPERIMENT IN ENVIRONMENT E1

Reservoir	n_i	n_o	n_r	α	d_t	$\mathbf{W}_{\mathrm{inp}}^{\mathrm{res}}$
Localization	8	8	400	0.01	10	$\{\pm 1(30\%), 0(70\%)\}$
Navigation	19	2	400	1	5	$\{\pm 1(50\%), 0(50\%)\}$

an initial room to a target room (see Fig. 4(a) for an example) - there are 6 possible routes in environment E1. The datasets were downsampled by a factor of $d_t = 10$ and $d_t = 5$ respectively (values empirically chosen to give best performance), resulting in two datasets of 50.000 and 20.000 samples, respectively. As the these sampling rates are different from each other, signals from the localization reservoir ($\mathbf{y}_{\mathrm{cloc}}$ and $\mathbf{y}_{\mathrm{ploc}}$) are upsampled to the same sampling rate of the navigation reservoir before they are used as input to that module. A summary of the parameter configuration is given in Table I. Some of these parameters are described in Section II-A. In this table, the connections in $\mathbf{W}_{\mathrm{inp}}^{\mathrm{res}}$ are initialized to +1, -1 and 0 with probabilities 0.15, 0.15 and 0.7 (0.25, 0.25 and 0.5), respectively, for the localization reservoir (navigation reservoir). These settings for the connections are not crucial for the experiments in this work (they are usually chosen to be sparse). Parameters α and d_t were found by a grid search in the case of the localization module (offline testing), and empirically in the case of the navigation module (online testing).

The localization performance on test data (consisting of 50.000 samples downsampled to 5.000 timesteps) is shown in Fig. 5. It can correctly detect the current robot room 97.5% of the time and the previously visited room 97.8 % of the time (this result is consistent if different randomly generated reservoirs are considered). Examples of the successful trajectories generated by the RC-HC system after training are shown in Fig. 6. The robot starts in one of the rooms (position indicated by a circle) and navigates to the goal room (given as input) with the end position represented by a small cross. The trajectory is plotted such that its color changes from green to blue, representing the progress of the navigation. In Fig. III, it is shown that the trained system can easily recover from a kidnapping event. The robot started at room 1 and aimed at room 3 as a goal. After reaching room 3, its goal changed back to room 1, but few timesteps later it was kidnapped to room 2. It is possible to see that although it was displaced to another room, the robot drove successfully to its destination (goal room 1). This result is consistent across multiple trials and experiments. In 63 routes that were evaluated, the RC-HC controller could successfully drive the robot to the correct room in all cases without any collision. These results are summarized in Table II.

The second environment E2 has 9 rooms and only 4 of them will be used as starting and goal locations: rooms 1, 3, 7 and 9. In this way, starting in one of the 4 locations, there are 12 possible shortest (optimal) routes that the robot can follow. The training datasets are also generated in the same way as before, but now 500.000 samples represent only 32 routes, which are less examples for training than for environ-
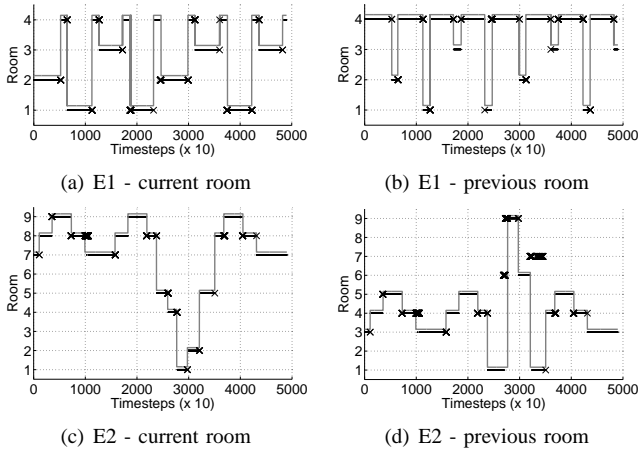
Fig. 5. Performance results of the localization module in environments E1 and E2. Predicted locations are represented by black points whereas solid grey lines are the true robot location. Black crosses represent mistakes.

ment E1. See Fig. 4(b) for an example of robot trajectories generated with the supervisor controller. The experiments in environment E2 use the same configuration stated for previous experiments except for the following changes. The number of outputs n_o of the localization module is 18 (9 previously visited rooms and 9 current rooms). The number of inputs n_i for the navigation reservoir is 30 (18 from the localization module + 4 goal inputs + 8 distance sensors).

The localization performance on test data for environment E2 is shown in Fig. 5(c). The system can detect the current and previously visited room 96.33% and 93.63% of the time, respectively. An example of successful trajectory in environment E2 is shown in Fig. 7(a). The robot, driven by the RC-HC controller, starts at room 1 and reaches room 7 successfully. In 15 out of 23 runs, the robot could perfectly follow the optimal (shortest) path to its goal. In all 23 runs it was able to complete the task. Task completion means that the robot reaches the goal location, being acceptable that during navigation it takes a wrong decision and then goes back to the correct optimal path (see Fig. 7(b) for an example). This also shows the robustness of the RC-HC controller to noise and unpredictable situations. A summary of the experimental results is given in Table II.

It is important to observe that most of the errors of the localization module are made at the transitions between one room and the following one. These errors represent a temporary confusion, which is better than a permanent mistake. Although navigation does not start in intermediate rooms in environment E2 during testing, it is expected that the robot can reach any goal location regardless of its initial position as long as the same sub-route appears during training. Generalization has been tested to the extent of the kidnapping event. Future work should confirm that

TABLE II

PERFORMANCE RESULTS IN NUMBER OF TRAJECTORIES

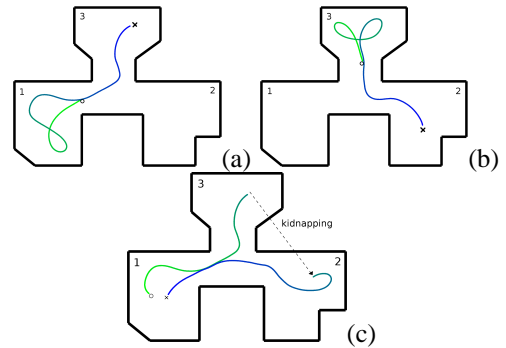|	Shortest Path	Task completion
Environment E1	63 out of 63 (100%)	100%
Environment E2.	15 out of 23 (65%).	100%

Fig. 6. Trajectories for robot driven by the RC-HC controller in environment E1. (a) Robot starts at room 1 and goes to room 3. (b) Robot starts at room 3 and goes to room 2. Starting and ending positions are marked with a circle and a cross, respectively. (c) The robot drives from room 1 to goal room 3. In room 3, its goal changes back to room 1, but it is kidnapped to room 2 after few timesteps. The trajectory shows that it recovered nicely from the kidnapping once it drove directly back to room 1.
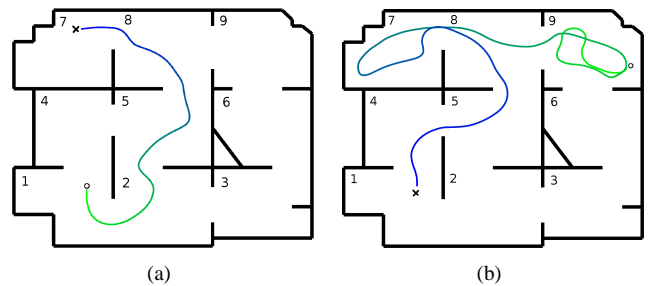

Fig. 7. Trajectories for robot driven by the RC-HC controller in environment E2. (a) Starting at room 1 and going to target room 7 via rooms ($2 \rightarrow 5 \rightarrow 8$) (optimal path). (b) Starting at room 9 and going to target room 1 via rooms ($8 \rightarrow 7 \rightarrow 8 \rightarrow 5 \rightarrow 4$) (task completion). Starting and ending positions are marked with a circle and a cross, respectively.

the trained system can avoid dynamic unseen obstacles during testing while reaching the desired goal locations. This generalization capability is expected to work with our proposed architecture once it has been shown that reservoir architectures can learn and generalize obstacle avoidance behaviors [24].

IV. CONCLUSIONS

This work proposes a hierarchical architecture based on a biologically plausible [26] technique for training Recurrent Neural Networks, the Reservoir Computing [18] approach. The RC-HC architecture constructs an internal model of the environment as it is trained by a series of examples generated by a supervisor controller (a program or a human supervisor). In this imitative setting, the architecture learns a cognitive, implicit map of the environment from a set of 8 low-accuracy distance sensors, which is used for goal-oriented navigation in simple and complex simulated environments.

The proposed RC-HC architecture has two reservoir modules, one for localization and another for navigation. The first module predicts the current room as well as the previously visited room. It was important to also learn to predict the previously visited room in order to boost the memory of the whole navigation system, so that the trained system had some sense of directionality (which room the robot came from) for making a correct route to the goal. The navigation module

integrates different sources of information such as from the distance sensors, the output of the localization module and the goal location, being able to produce behaviors which contain reactive (obstacle avoidance) and deliberative (decision making) components.

The RC training method exploits the capabilities of the reservoir to project its inputs to a high-dimensional feature space, where it is easier to separate and classify (dynamic) patterns existent in the environment. In this way, the reservoir states (with its rich dynamics) are simply linearly combined to predict the desired output of the system, be it either the robot location or the desired motor actuators. This mapping between reservoir states and the desired output is the only part necessary to be trained, usually through linear regression methods. So, avoiding training the recurrent reservoir itself also avoids problems with convergence of the training process (as it happens with BPTT method).

The proposed architecture works with distinct timescales for agile processing of low-level sensory-motor behaviors as well as for slow processing of higher-level concepts such as locations. This is achieved by having two reservoirs working with distinct leak rates, each one responsible for the respective skill, localization (*slow* timescale) or navigation (*fast* timescale) (relevant works such as [25] also elaborate on a hierarchy of slow and fast networks for humanoid robot skill learning).

The current method requires no special landmarks to be placed in the environment and works with cheap small mobile robots having few noisy infra-red distance sensors. Although the environment rooms appear to be different in shape from each other, it has been show that the localization performance is not deteriorated if the environment has multiple symmetric rooms [2]. Future directions for research include the study of integrating rich visual data (from a camera) for helping navigation in a complex human environment, probably using bigger robots. In this context, high-dimensional multimodal input should be pre-processed and reduced to fewer dimensions before feeding it to a reservoir. Experiments with the real e-puck robot are also planned as future work, as a way of confirming the robustness of the method with respect to parameter tuning. This work provides an imitation-based paradigm for learning autonomous navigation capabilities, but an interesting point for research is to let the robotic system interact with the environment and learn in on-line way to map the environment as well as to reach goal locations based on rewards given by the environment.

V. ACKNOWLEDGMENTS

REFERENCES

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

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

[3] 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.

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

[5] T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part ii state of the art. *Robotics and Automation Magazine*, pages 108–117, September 2006.

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

[7] R. Chavarriaga, T. Strsslin, D. Sheynikhovich, and W. Gerstner. A computational model of parallel navigation systems in rodents. *Neuroinformatics*, 3:223–241, 2005.

[8] H. 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.

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

[10] P. Joshi and W. Maass. Movement generation with circuits of spiking neurons. *Neural Computation*, 17(8):1715–1738, 2005.

[11] 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.

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

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

[14] F. Mondada. E-puck education robot, September 2007. http://www.e-puck.org/.

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

[16] P. G. Plöger, A. Arghir, T. Günther, and R. Hosseiny. Echo state networks for mobile robot modeling and control. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 157–168, 2004.

[17] M. Salmen and P. G. Plöger. Echo state networks used for motor control. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1953–1958, 2005.

[18] 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.

[19] D. Silver, J. A. D. Bagnell, and A. T. Stentz. Perceptual interpretation for autonomous navigation through dynamic imitation learning. In *International Symposium of Robotics Research*, August 2009.

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

[21] J. Tani. On the interactions between top-down anticipation and bottom-up regression. *Frontiers in Neurorobotics*, 1, 2007.

[22] T. van der Zant, V. Becanovic, K. Ishii, H. Kobialka, and P. Plöger. Finding good echo state networks to control an underwater robot using evolutionary computations. In *Proceedings of the 5th IFAC symposium on Intelligent Autonomous Vehicles (IAV04)*, 2004.

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

[24] T. Waegeman, E. Antonelo, F. wyffels, and B. Schrauwen. Modular reservoir computing networks for imitation learning of multiple robot behaviors. In *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, 2009.

[25] Y. Yamashita and J. Tani. Emergence of functional hierachy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS Comput Biol*, 4(11):e1000220, 11 2008.

[26] T. Yamazaki and S. Tanaka. The cerebellum as a liquid state machine. *Neural Networks*, 20:290–297, 2007.