Open access • Journal Article • DOI:10.1109/TNNLS.2014.2323247

On Learning Navigation Behaviors for Small Mobile Robots With Reservoir Computing Architectures — Source link

Eric Aislan Antonelo, Benjamin Schrauwen

Institutions: Universidade Federal de Santa Catarina, Ghent University

Published on: 01 Apr 2015 - IEEE Transactions on Neural Networks (IEEE)

Topics: Reservoir computing, Mobile robot, Behavior-based robotics and Recurrent neural network

On Learning Navigation Behaviors for Small Mobile Robots with Reservoir Computing Architectures

Eric Aislan Antonelo and Benjamin Schrauwen

***Abstract*—This work proposes a general Reservoir Computing (RC) learning framework which can be used to learn navigation behaviors for mobile robots in simple and complex unknown, partially observable environments. RC provides an efficient way to train recurrent neural networks by letting the recurrent part of the network (called *reservoir*) fixed while only a linear readout output layer is trained. The proposed RC framework builds upon the notion of *navigation attractor* or behavior which can be embedded in the high-dimensional space of the reservoir after learning. The learning of multiple behaviors is possible because the dynamic robot behavior, consisting of a sensory-motor sequence, can be linearly discriminated in the high-dimensional nonlinear space of the dynamic reservoir. Three learning approaches for navigation behaviors are shown in this paper. The first approach learns multiple behaviors based on examples of navigation behaviors generated by a supervisor, while the second approach learns goal-directed navigation behaviors based only on rewards. The third approach learns complex goal-directed behaviors, in a supervised way, using an hierarchical architecture whose internal predictions of contextual switches guide the sequence of basic navigation behaviors towards the goal.**

***Index Terms*—robot navigation, reservoir computing, reinforcement learning, goal-directed navigation, recurrent neural networks, echo state network, sensory-motor coupling.**

I. Introduction

BEHAVIOR-BASED approaches to robotics have been proposed early in the literature [1], [2]. Instead of having several modules for perception, world modeling, planning and execution, they are based on individual intelligent control modules, where each one contributes to behavior generation for controlling a robot, thus following a bottom-up approach.

This work aims at designing intelligent navigation systems from a bottom-up perspective, where learning of implicit world representations and complex sensory-motor coupling is inspired by the implicit, basic mechanisms of intelligence which control biological systems. Thus, an essential requirement is that these intelligent systems process information and become *situated* in the environment [3] by solely using their local view of the environment given by the sensory apparatus present in the agent or robot. This embodiment of the robot implies that its control architecture should possess an internal state which represents its perceptual history of the world.

Recurrent Neural Networks (RNNs) are a good candidate for that since they have an internal state made possible by the network's recurrent connections. However, traditional training for RNNs, such as Backpropagation through time [4], has slow convergence and does not guarantee to find the global optimum.

Training for RNNs is much simplified and efficiently executed under the recently emerging paradigm of Reservoir Computing (RC) [5] (see Figure 3). This is because the recurrent non-linear part of the network (called reservoir) is left fixed, while only a linear output layer is trained, usually through standard linear regression techniques. This type of *state-dependent computation* has been proposed as a biologically plausible model for cortical processing [6], [7], [8]. Such theoretical models include: *Echo State Networks* (ESN) [9] for analog neurons and *Liquid State Machines* (LSM) [7] for spiking neurons. Many applications of RC exist: online adaptive control of robotic arms [10], [11], optoeletronic applications [12], speech recognition [13], etc. From a machine learning perspective, a reservoir network, usually randomly generated and sparsely connected, functions as a *temporal kernel* [14], projecting the input to a dynamic high-dimensional space. During simulation, the reservoir states form a trajectory which is dependent on the current external sensory input, but which still contains memory traces of previous stimuli. Computation in the output layer occurs by linearly reading out instantaneous states of the reservoir. In this way, reservoir architectures can inherently process spatiotemporal patterns.

In this work, navigation behaviors are modeled using the RC paradigm, where ESNs serve as a general mechanism to build embodied mobile robots with an internal environmental representation. Additionally, they are designed according to the notion of navigation attractor[1]. A navigation attractor (Fig. 1) is a *reactive robot behavior* defined by a spatiotemporal pattern resulting from a specific sensory-motor coupling which a mobile robot can execute in its environment. Under this scheme, a robot tends to follow a trajectory with attractor-like characteristics in space. These navigation attractors are characterized by being robust to noise and unpredictable events

E. A. Antonelo is with the Department of Automation and Systems, Federal University of Santa Catarina, Brazil and B. Schrauwen is with the Department of Electronics and Information Systems, Ghent University, Belgium, e-mail: erantone@elis.ugent.be

E. Antonelo acknowledges the Universiteit Ghent (Belgium), the National Council for Scientific and Technological Development (CNPq-Brazil) and the National Council for the Improvement of Higher Education (CAPES-Brazil) for their financial support.

This document is a draft version of the paper. The published version can be found in the IEEE Transactions on Neural Networks and Learning Systems.

[1]The term *attractor* is used in this paper more metaphorically and does not directly relate to the exact definition of attractor in mathematics.

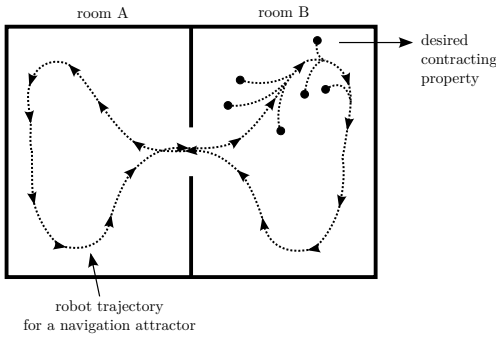


Fig. 1. Representation of a reactive navigation attractor or behavior in the environment space and desired contracting property.

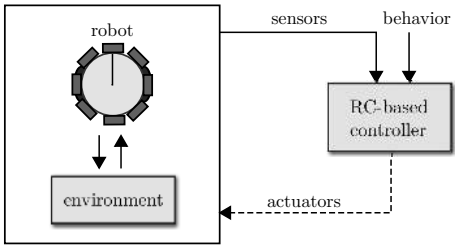


Fig. 2. Modeling multiple reactive behaviors or navigation attractors using a single RC network via external binary input channel. Dashed connections are trainable.

and by having inherent collision avoidance skills. In this work, it is shown that an RC network can model not only one behavior, but multiple navigation behaviors by shifting the operating point of the dynamical reservoir system into different *sub-space attractors* using additional external inputs representing the selected behavior. The sub-space attractors emerge from the coupling existing between the RC network, which controls the autonomous robot, and the environment (Fig.2).

This paper presents three approaches for learning complex robot behaviors following the idea of embedding sub-space attractors, corresponding to reactive behaviors, into the dynamic reservoir space[2]. The first approach uses a single reservoir for learning behaviors in a supervised way [15], that is, by showing examples of two different navigation behaviors to the network. An external binary input selects the behavior which the network should reproduce. After training, the RC network is able to replicate and switch between these different behaviors, where each one corresponds to a different sub-space attractor in the reservoir state space. The settings, experiments and results corresponding to the first approach are shown in Section IV.

The second approach uses a reinforcement learning framework to shape navigation behaviors through trial and error [16]. The reward is given only at the destination location, while the correct path of the robot to the goal is dependent on a temporary initial stimulus, which make the environment partially observable. It is shown that the recurrent weights of the network are an important feature for partially observable environments, since they provide a transient memory for these types of delayed response tasks. The settings, experiments and results corresponding to the second approach are shown in Section V.

The third approach extends the first network to a hierarchical architecture which can autonomously switch between different contexts and select the appropriate behavior according to the predicted context [17]. This is achieved by training one network to predict the current robot location in a multi-room environment (localization reservoir) [18], and another network to drive the robot through the environment (navigation reservoir). The goal location is given as input as well as the distance sensors of the robot. After training the architecture with examples of trajectories from a starting room to a goal room in the multi-room environment, the navigation reservoir can generate a sequence of reactive behaviors towards the goal. The settings, experiments and results corresponding to the third approach are shown in Section V.

This work shows that it is possible to learn complex navigation behaviors, either by reinforcement learning (where the behavior improves progressively by interaction with the environment) or by supervised learning (where behaviors are embedded through one shot training process with desired sensory-motor coupling), using a recurrent neural network model, the Echo State Network (RC network). It also shows that contextual switches (elicited by entering another room in an environment) predicted by a RC network in a hierarchical architecture can be used to generate an autonomous sequence of reactive behaviors for goal-directed navigation.

While feedforward networks with time-windowed inputs can show good results in a variety of temporal tasks, they do not satisfy some of our requirements: are not biologically plausible; can not generate implicit internal representations based on dynamical states (as RNNs do); and their iterative training process hinders the concrete realization of the control task as it will be seen in this paper.

In the next section, a short review on biologically-inspired navigation systems as well as a comparison with the proposed approaches in this paper are presented. Section III presents the basic Reservoir Computing model used in this work (Section III-A), the robot models used in the experiments (Section III-B), and the concept of navigation attractors corresponding to the robot behaviors (Section III-C).

II. Related work on biologically-inspired navigation systems

There are several works in the literature which employ Recurrent Neural Networks (RNNs) for designing localization and navigation systems for mobile robots. In [19], RNNs are used for model-based learning in robot navigation. In order to achieve situatedness during navigation, a forward model of the mobile robot is learned in a self-organized way using Backpropagation through time. The internal model predicts the next sensory input given the current sensors (range image and travel distance) and the motor output. In this way, it learns to be situated through interaction with the environment by learning the environmental attractor in the offline training

[2]Preliminary results on the approaches presented here were already published in conference proceedings [15], [16], [17] .

phase. Other early related works for situated robotics are [20] and more recently [21].

In [22], evolutionary strategies for RNNs are tackled in the context of a homing navigation task. In their work, a RNN is evolved so that a mobile robot drives as long as possible around an arena and goes back to a recharging area whenever its battery level is near empty. The evolved RNN learned an internal representation which is a function of the robot position and of the battery level.

Other models of hippocampal place cells and biologically-inspired navigation exist in the literature. In [23], unsupervised growing networks are used to build an architecture with idiothetic and allothetic components that are combined in a hippocampal place cell layer to support spatial navigation (validated using a Khepera mobile robot with 2D vision sensors). Their model explicitly uses dead-reckoning to track the robot position and associates place cell firing with the estimated position.

In [24], a hippocampal place cell model is designed to solve the SLAM problem. They choose a pragmatic approach, favoring functionality over biologically plausibility. Their model, called RatSLAM, has a 3D structure for pose cells (representing beliefs for the robot position and orientation) which learn associative connections with view cells (allothetic representation). They validate their model with several mobile robots, equipped with a camera, in indoor and outdoor environments. Other works oriented towards modeling an animal's capability for spatial navigation are given in [25], [26]. A single learning technique which maximizes slowness of the output signal applied to hierarchical networks is able to generate self-organized representations of place cells as well as of head-direction cells [27] without odometry information. A similar method based on temporal stability for learning hippocampal place cells for mobile robots is given in [28]. For a further (and older) review on biologically-inspired localization models, see [29] and [30].

In [31], an ESN is used to model behavior acquisition by demonstration for a Khepera mobile robot using an 8x6 color image as input to the network. They train the ESN to perform a sequence of reactive behaviors (find and reach target), which actually do not require the dynamic properties of the reservoir since their results show that the same performance can be achieved if the recurrent connections from the reservoir are removed. The work presented in this paper goes beyond in three ways: we build upon the idea of dynamic sub-space attractors in the reservoir state space for embedding multiple navigation behaviors; for acquiring increasingly complex behaviors, hierarchical architectures are built which handle context room switches; and it is shown that the same RC architecture can be used in a reinforcement learning task.

Most of the aforementioned models are based on rich visual (pixel-based) stimuli as external sensory input and/or use odometry for path integration. In contrast to this, the RC-based navigation systems in this work are based solely on low-dimensional input such as few infra-red distance sensors (apart from the first approach in Section IV which uses a few more *color sensors*). Thus, the models presented here make no use of odometry for position estimation, even though we may predict the robot location using an additional RC network (third approach in Section VI). This work also relies on the fact that the ambiguous perceptual input space of the robot is disambiguated in the dynamical high-dimensional space of the reservoir, making it possible to distinguish similar locations from the robot's perspective. This is possible because of the fading memory characteristic of RC networks. A third main prominent feature of our approach is that the reservoir, functioning as a temporal non-linear kernel [14], can be used in supervised, unsupervised and reinforcement learning tasks by only changing the training method in the linear output layer, characterizing it as a multi-faceted machine learning method [32].

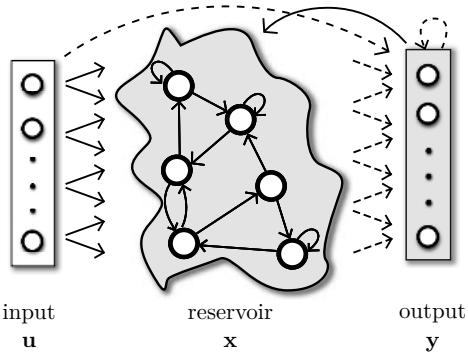


Fig. 3. Reservoir Computing (RC) network. The reservoir is a non-linear dynamical system usually composed of recurrent sigmoid units. Solid lines represent fixed, randomly generated connections, while dashed lines represent trainable or adaptive weights.

III. Methods

A. Reservoir Computing

1) ESN model: An ESN is composed of a discrete hyperbolic-tangent RNN, the reservoir, and of a linear readout output layer which maps the reservoir states to the actual output. Let n_i, n_r, n_o represent the number of input, reservoir and output units, respectively, $\mathbf{u}[n]$ the n_i-dimensional external input, $\mathbf{x}[n]$ the n_r-dimensional reservoir activation state, $\mathbf{y}[n]$ the n_o-dimensional output vector. Then the discrete time dynamics of the ESN is given by the state update equation

$$\begin{aligned}\mathbf{x}[n+1] =&(1-\alpha)\mathbf{x}[n]+\alpha f(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}[n]+\mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}[n]+\\ &\mathbf{W}^{\mathrm{r}}_{\mathrm{o}}\mathbf{y}[n]+\mathbf{W}^{\mathrm{r}}_{\mathrm{b}}),\end{aligned} \tag{1}$$

where: α is the leak rate [33], [34]; $f() = \tanh()$ is the hyperbolic tangent activation function, commonly used for ESNs, and by the output computed as:

$$\begin{aligned}\mathbf{y}[n+1] &= g\left(\mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{x}[n+1]+\mathbf{W}^{\mathrm{o}}_{\mathrm{i}}\mathbf{u}[n]+\mathbf{W}^{\mathrm{o}}_{\mathrm{o}}\mathbf{y}[n]+\mathbf{W}^{\mathrm{o}}_{\mathrm{b}}\right) &(2)\\ &= g\left(\mathbf{W}^{\mathrm{out}}\left(\mathbf{x}[n+1],\mathbf{u}[n],\mathbf{y}[n],1\right)\right) &(3)\\ &= g\left(\mathbf{W}^{\mathrm{out}}\mathbf{z}[n+1]\right), &(4)\end{aligned}$$

where: g is a post-processing activation function; $\mathbf{W}^{\mathrm{out}}$ is the column-wise concatenation of $\mathbf{W}^{\mathrm{o}}_{\mathrm{r}}$, $\mathbf{W}^{\mathrm{o}}_{\mathrm{i}}$, $\mathbf{W}^{\mathrm{o}}_{\mathrm{o}}$ and $\mathbf{W}^{\mathrm{o}}_{\mathrm{b}}$; and $\mathbf{z}[n+1] = (\mathbf{x}[n+1],\mathbf{u}[n],\mathbf{y}[n],1)$ is the extended reservoir state, i.e., the concatenation of the state, the previous input and output vectors and a bias term, respectively.

The matrices $\mathbf{W}^{\text{to}}_{\text{from}}$ represent the connection weights between the nodes of the complete network, where r,i,o,b denotes *reservoir*, *input*, *output*, and *bias*, respectively. All weight matrices representing the connections to the reservoir, denoted as $\mathbf{W}^{\text{r}}_{\cdot}$, are initialized randomly (represented by solid arrows in Figure 3), whereas all connections to the output layer, denoted as $\mathbf{W}^{\text{o}}_{\cdot}$, are trained (represented by dashed arrows in Figure 3).

Output feedback given by the projection $\mathbf{W}^{\text{r}}_{\text{o}}\mathbf{y}[n]$ and bias $\mathbf{W}_{\text{b}}$ are optional. In the absence of these terms, (1) and (2) become:

$$\mathbf{x}[n+1] = f\left(\mathbf{W}^{\text{r}}_{\text{r}}\mathbf{x}[n] + \mathbf{W}^{\text{r}}_{\text{i}}\mathbf{u}[n]\right) \tag{5}$$

$$\mathbf{y}[n+1] = g\left(\mathbf{W}_{\text{o}}\mathbf{x}[n+1]\right). \tag{6}$$

There are two ways to increase the memory of a reservoir which has no output feedback. It is possible to either tune the leak rate $\alpha \in (0,1]$ of the reservoir for matching the timescale of the input signal or downsample the input signal. Low leak rates yield reservoirs with more memory which can *remember* the previous stimuli for longer time spans. On the other hand, leak rates close to 1 are suitable for high-frequency input signals which vary in a faster timescale.

Next, the procedures for reservoir creation and dynamics tuning are presented. The non-trainable connection matrices $\mathbf{W}^{\text{r}}_{\text{r}}, \mathbf{W}^{\text{r}}_{\text{i}}, \mathbf{W}^{\text{r}}_{\text{o}}, \mathbf{W}^{\text{r}}_{\text{b}}$ are usually generated from a random distribution, such as a Gaussian distribution $N(0,1)$ or a uniform discrete set $\{-1,1\}$. During this initialization, two parameters are used:

- the connection fraction $c^{\text{to}}_{\text{from}}$ corresponds to the percentage of nonzero weights in the respective connection matrix $\mathbf{W}^{\text{to}}_{\text{from}}$.
- $\upsilon^{\text{to}}_{\text{from}}$ corresponds to the scaling of the respective connection matrix $\mathbf{W}^{\text{to}}_{\text{from}}$.

While the connectivity between units in $\mathbf{W}^{\text{r}}_{\text{i}}$ and $\mathbf{W}^{\text{r}}_{\text{r}}$ is not that important [35], although they are usually created considering sparse connectivity, the **scaling** of these matrices have a great influence on the reservoir dynamics [5] and must be tuned for achieving optimal performance.

The randomly generated $\mathbf{W}^{\text{r}}_{\text{r}}$ must be rescaled such that the dynamical system is stable but it still exhibits rich dynamics. As the ESN is usually nonlinear, this can be achieved by studying a linearized version of the ESN around the equilibrium point [36]. Under this assumption, a necessary condition to guarantee the **Echo State Property** (ESP) [37] for ESNs, i.e., a reservoir with fading memory[3], is to rescale $\mathbf{W}^{\text{r}}_{\text{r}}$ such that the maximal singular value of $\mathbf{W}^{\text{r}}_{\text{r}}$ is smaller than unity.

However, using the maximal singular value to rescale the reservoir connection matrix usually does not provide rich reservoir dynamics. An alternative is to rescale $\mathbf{W}^{\text{r}}_{\text{r}}$ such that its spectral radius $\rho(\mathbf{W}^{\text{r}}_{\text{r}}) < 1$ [37]. Although it does not guarantee the ESP, in practice it has been empirically observed that this criterium works well and often produces analog sigmoid ESNs with ESP for any input, producing richer reservoirs which contain signals with multiple frequencies. For most applications, the best performance is attained with a reservoir that operates at the **edge of stability**, e.g., $\rho(\mathbf{W}^{\text{r}}_{\text{r}}) = 0.99$.

Considering a normalized input signal $\mathbf{u}[n]$, the effect of input scaling υ^{r}_{i} on the reservoir dynamics is such that, the larger the scaling, the closer to saturation the reservoir states will be, since the reservoir state is shifted torwards the non-linear area of the tanh activation function. Spectral radius closer to unity as well as larger input scaling makes the reservoir more non-linear, which has a deterioration impact on the memory capacity as side-effect [38].

The scaling of these non-trainable weights is a parameter which should be chosen according to the task at hand empirically, analyzing the behavior of the reservoir state over time, or by grid searching over parameter ranges.

Although it is suggested that many parameters should be optimized, RC is quite robust to several of these parameters. Thus, it is relevant to mention the two most important parameters for tuning in this work: **leak rate** (or, alternatively, resampling rate of the input signal) and **input scaling**. The other parameters are less important.

2) Readout Output Training: The readout output of the RC network is the only layer to be trained, usually by standard **linear regression** methods. For that, the reservoir is driven by an input sequence $\mathbf{u}(1),\ldots,\mathbf{u}(n_s)$ which yields a sequence of extended reservoir states $\mathbf{z}(1),\ldots,\mathbf{z}(n_s)$ using (1).

The desired teacher outputs $\hat{\mathbf{y}}[n]$ are collected row-wise into a matrix $\hat{\mathbf{Y}}$. The generated extended states are collected row-wise into a matrix $\mathbf{X}$ of size $n_s \times (n_r + n_i + 1)$ if no output feedback is used. Then the training of the output layer consists of finding the weights $\mathbf{W}^{\text{out}}$ which minimizes the sum of squared errors

$$\sum_{t=1}^{n_s} \left(\hat{\mathbf{y}}[n] - \mathbf{y}[n]\right)^2, \tag{7}$$

by using the Moore-Penrose generalized matrix inverse[4], or *pseudo-inverse* $\mathbf{X}^{\dagger}$ of matrix $\mathbf{X}$:

$$\mathbf{W}^{\text{out}} = \mathbf{X}^{\dagger}\hat{\mathbf{Y}} = (\mathbf{X}^{\top}\mathbf{X})^{-1}\mathbf{X}^{\top}\hat{\mathbf{Y}} \tag{8}$$

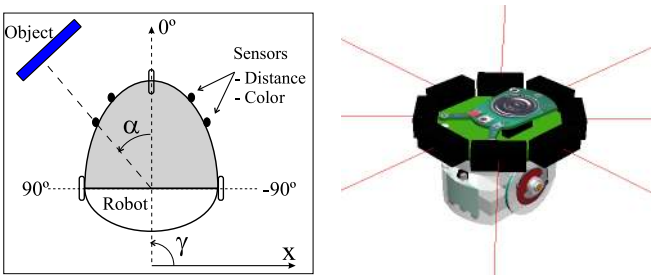
where n_s denotes the total number of training samples and the initial state is $\mathbf{x}(0) = \mathbf{0}$. Note that the other matrices $(\mathbf{W}^{\text{r}}_{\text{r}}, \mathbf{W}^{\text{r}}_{\text{i}}, \mathbf{W}^{\text{r}}_{\text{b}}, \mathbf{W}^{\text{r}}_{\text{o}})$ are not trained at all.

It is important to note that there is an initial transient during the generation of reservoir states $\mathbf{x}[n]$ using (1) due to the fading memory of the reservoir, which may be undesired for the readout training. So, the usual procedure to deal with this is to disregard the first n_{wd} samples in a process called **warm-up drop** so that only the samples $\mathbf{z}[n], n = n_{wd}, n_{wd}+1, ..., n_s$ are collected into the matrix $\mathbf{X}$. Although this procedure is always used in this work, the notation for the generation of reservoir states will not change for the sake of simplicity.

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).

[3]The Echo State Property states conditions for the ESN principle to work. It can be understood as having a reservoir with fading memory which asymptotically washes out any information from initial conditions.

[4]For numerical stability, we may also employ QR decomposition.



(a) SINAR (b) Extended Webots e-puck

Fig. 4. Robot models. (a) SINAR robot model with distance and color sensors (usually in number of 17 of each type) positioned in the frontal part of the robot (−90 to 90). (b) *Modified e-puck robot* from Webots simulation environment, **extended** with simulated longer-range infra-red sensors capable of reading distances from 5 cm to 80 cm (modeling a real infra-red sensor).

3) Error measure: For regression tasks, the Normalized Mean Square Error (NMSE) is used as a performance measure and is defined as:

$$\text{NMSE} = \frac{\langle (\hat{y}[n] - y[n])^2 \rangle}{\sigma^2_{\hat{y}[n]}}, \tag{9}$$

where the numerator is the mean squared error of the output $y[n]$ and the denominator is the variance of desired output $\hat{y}[n]$.

B. Robot Models

1) SINAR: SINAR is a 2D autonomous robot simulator introduced in [39], where the mobile robot (Fig. 4(a)) interacts with the environment by distance and color sensors; and by one actuator which controls the movement direction (turning).

The environment of the robot is composed of several objects, each one of a particular color. Particularly, obstacles are represented by blue objects whereas targets are given by yellow objects. The robot model has 17 sensor positions distributed uniformly over the front of the robot, from -90° to +90°. Each position holds two virtual sensors for distance and color perception. The distance sensors are limited in range such that they saturate for distances greater than 300 distance units ($d.u.$), and are noisy - they exhibit Gaussian noise $N(0, 0.01)$ on their readings. 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 equal to 0.28 distance units ($d.u.$)/s.

2) E-puck: The e-puck [40] is a small differential wheeled robot which was built primarily for education purposes, but has been largely adopted in research as well. The mobile robot is equipped with 8 infra-red sensors which measure ambient light and proximity of obstacles in a range of $[0-4]cm$ originally, which effectively restricts the ability to read distances to obstacles. The actuators of the robot are 2 stepper motors.

The *variant robot model* used in this work is the simulated e-puck extended with 8 infra-red sensors which can measure distances in the range [5-80] cm ([0-80] cm for the reinforcement learning task). The original simulation model of the e-puck has a 5.20 cm diameter, but it increases to 10 cm when modified with the extra turret for the infra-red sensors. The speed of the robot is limited to the interval ±[0, 300] steps/s (or ±[0, 3.77] cm/s).

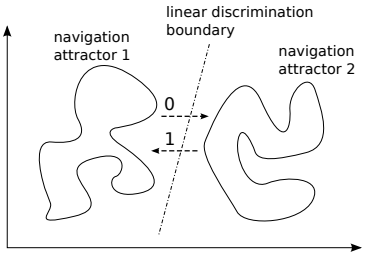


Fig. 5. Example of two navigation attractors in a hypothetical bi-dimensional dynamical system space. Dashed arrows represent switching events caused by activities of external input channels.

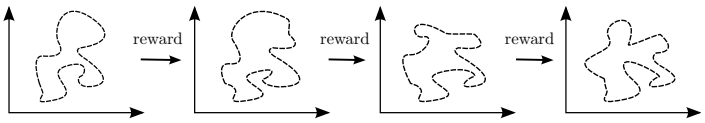


Fig. 6. Reinforcement Learning shapes navigation attractor in a hypothetical bi-dimensional dynamical system space as learning evolves. The attractor is dynamic, i.e., changes over time with learning iterations.

C. Sub-space attractors in high-dimensional space

For empowering navigation systems with a more complex and high-level behavior, it is necessary to simultaneously learn multiple reactive navigation attractors.

In order to embed multiple reactive behaviors into a single RC network, it is necessary to add external binary inputs to the RC network (Fig. 2), capable of shifting the attractor dynamics to a confined sub-space corresponding to the selected behavior. The external input acts as a bias during the execution of a reactive behavior. A switch to a different behavior will cause a shift into a different operating point of the reservoir, which in turn is coupled to the environment.

As this architecture (Fig. 2) is trained using linear regression on the dynamical system space (only the motor actuators given by the dashed connections are trained), the shift in the high-dimensional space caused by the external binary input makes possible that a linear discrimination is sufficient to confine navigation attractors to different sub-spaces (Fig. 5). Thus, this architecture supports the simultaneous learning of many (even conflicting) behaviors by the trick of shifting the reservoir state space. The number of behaviors that could be learned is limited by the memory capacity of the network [41].

A second approach to learn navigation attractors is through reinforcement learning (RL). Under this scheme, the RC network does not receive a teacher signal, but only a reward signal usually indicating success or failure. Thus, learning is achieved by trial and error, which means that a lot of random trials will take place in the beginning of the learning process. During this iterative learning procedure, the navigation attractor learned by the RNN is actually dynamic, i.e., changing over time (Fig. 6).

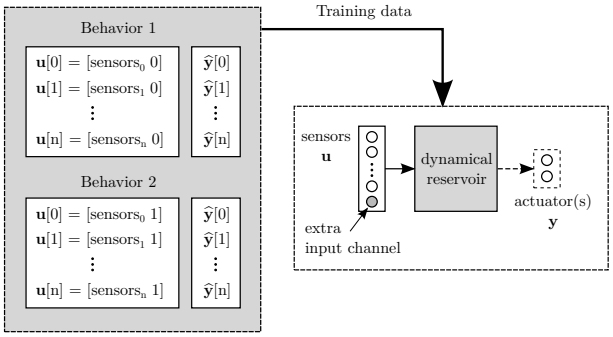


Fig. 7. Training a single RC network for learning 2 different behaviors. Behaviors 1 and 2 are generated by distinct teacher controllers. The input **u** is the concatenation of the sensors and an extra input channel (0 or 1) or binary vector for behavior selection.

IV. First approach: Supervised learning of navigation behaviors

The first approach for modeling autonomous navigation systems for small mobile robots in this paper is by imitation learning of robust reactive behaviors. By taking this approach, learning is accomplished by generating examples of the desired sensory-motor coupling using a supervisor or teacher controller.

After the learning process, the coupling of the dynamical system (reservoir), which controls the robot, and the environment allows that the robot becomes situated in its environment since the internal state of the reservoir reflects the contextual state of the environment.

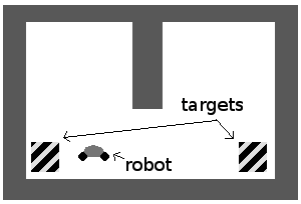
A. Training the Reservoir Architecture with Examples

The robot model used in this section is the SINAR model, described in Section III-B1.

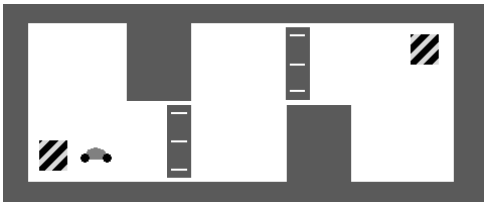
The samples generated by teacher (supervisor) controllers containing data from distance and color sensors, and from actuators are used to train the RC-based controller in a Matlab environment. The experimental setup is given in the following section.

The imitation learning procedure, depicted in Fig. 7, can be summarized in four stages:

1) First, the teacher controllers navigate in a particular environment, e.g., avoid obstacles and/or seek targets.
2) In a second stage, data samples with the observed sensory-motor couplings are collected from the teacher controllers during a robot run of a specific duration.
3) If there are multiple behaviors possibly from different controllers, the third stage concatenates the data collected in the previous stage, and adds extra binary input channel(s) for behavior selection (where each possible binary value could correspond to a behavior, e.g., 01, 10, and 11 encode three different behaviors).
4) The fourth stage corresponds to training the RC-based controller with the data collected in the second stage and concatenated in the third stage by supervised learning methods such as linear regression (Section III-A2).



(a) S1



(b) S2

Fig. 8. 2D environments used for the experiments in this section. 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 or present. (a) Small environment with two targets and one robot. (b) Big environment with a robot, two targets and two dynamic *blinking* obstacles (marked with three white stripes) which block the robot's way by appearing at random times during simulation.

B. Experiments

In this section, an RC network is trained to reproduce the following combined 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 as well as avoid obstacles.

The environments used for the experiments are shown in Fig. 8. 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 navigates through the environment normally performing cyclic trajectories. Captured targets are sequentially put back in the same locations after a capture[5]. Fig. 9 shows examples of navigation trajectories.

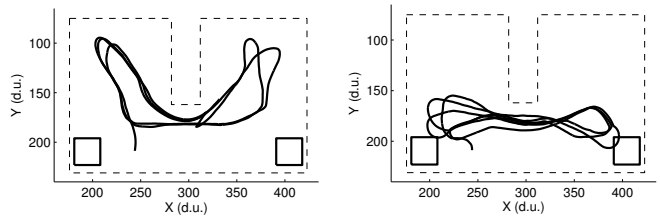
As EE and TS behaviors are conflicting behaviors, they must be generated by different teacher controllers. In the following, it is explained how these controllers are constructed using the intelligent navigation system described in [39].

EE The teacher controller which implements the EE behavior is trained to **avoid** blue objects (obstacles) and yellow objects (targets). An example of exploratory behavior which **ignores targets** is given in Fig. 9(a).

TS The teacher controller that generates the TS behavior is trained to **avoid** blue objects (obstacles) and to **seek** yellow objects (targets). The resulting target seeking behavior is shown in Fig. 9(b).

Next, the samples with sensory and actuator information are collected from teacher controllers in two stages. In the **first stage**, the controller implementing EE behavior steers the robot in environment S1 from Fig. 8, exploring the environment and ignoring targets. All sensory inputs and actuators

[5]A target capture causes the removal of the respective target from the environment.



(a) EE (b) TS

Fig. 9. Example of navigation trajectories of teacher controllers in environment S1. (a) EE exploratory behavior (ignores visible targets). (b) TS target seeking behavior (continually captures targets).

TABLE I
PARAMETER CONFIGURATION FOR RC-BASED CONTROLLER

Number of input channels	$n_i = 35$
Input connection fraction	$c_i^r = 0.2$
Input scaling	$\upsilon_i^r = 0.2$
Input downsampling	$d_t = 1$
Input to output connections	yes
Bias connection fraction	$c_b^r = 1$
Bias scaling	$\upsilon_b^r = 0.8$
Reservoir size	$n_r = 600$
Reservoir connection fraction	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_\mathrm{r}^\mathrm{r}) = 0.9$
Leak rate	$\alpha = 1$
Number of output channels	$n_o = 1$
Output feedback to reservoir	no

are recorded. In the **second stage**, the controller with TS behavior steers the robot in the same environment, but now generating a different trajectory towards the targets. Each stage lasts 22,500 timesteps, summing up 45,000 timesteps in total which corresponds approximately to 24 cyclic trajectories or loops in the respective environment.

After collecting the training data which represent EE and TS behaviors individually, a single RC network is trained to reproduce both behaviors by means of concatenation of the the training data as well as of an extra input channel added for behavior selection, as described in previous section and in Fig. 7. If this extra input has value zero (one), then the EE (TS) behavior is selected.

C. Settings

The parameter configuration for the RC network which controls the robot is shown in Table I. The inputs to the network are 17 distance sensors, 17 color sensors, plus 1 input for behavior selection, summing up $n_i = 35$ inputs. The reservoir size is $n_r = 600$ neurons. The output unit corresponds to the turning or direction adjustment robot actuator (the robot has constant velocity). The training is done according to Section III-A2 using the collected data of 45,000 timesteps, of which half of the observations has the value of the extra input channel set to 0 for EE behavior, and the other half has this value set to 1 for TS behavior.

The optimization of the spectral radius $\rho(\mathbf{W}_\mathrm{r}^\mathrm{r})$ for each experiment in this work was not necessary because the changes in performance were insignificant. Thus, setting the spectral radius at the edge of stability ($\rho(\mathbf{W}_\mathrm{r}^\mathrm{r}) \in [0.9, 1)$) has yielded very good results. Additionally, the specific setting of the input weight matrices ($c_i^r, c_b^r, \upsilon_i^r, \upsilon_b^r$) is not particularly critical for the experiments, allowing for other parameter ranges (although it could still be optimized).

D. Results

After learning in environment S1, the RC-based controller was evaluated in environments S1 and S2. The results for environment S1 are shown in Fig. 10. The simulation takes 20,000 timesteps. At each period of 5,000 timesteps, a behavior switching event takes place. Fig. 10(a) shows the coordinates of the robot during the run, where vertical lines represent the moments in which a behavior switching occurs. It can be seen that 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. 10(b) shows the corresponding robot trajectory in a 2D map during the simulation. The black (gray) trajectory corresponds to the time interval in which the EE (TS) behavior is selected.

From these figures, it can be observed that the trajectories form navigation attractors in the environment. In addition, switching between these attractors is accomplished smoothly by the RC-based controller without collisions to obstacles.

By reducing the high-dimensional state space of the dynamical reservoir, using Principal Component Analysis (PCA) on the reservoir states, it is possible to observe that *sub-space attractors* which are linearly separable (Fig. 11). By only changing an input from 0 to 1 or vice-versa, the operating point of the dynamical reservoir is changed to a different *sub-space attractor* in the dynamical system space, defined by the tight coupling between robot controller and environment.

Table II shows results for different number of neurons (n_r) in the reservoir. Each row shows the mean values of the: training NMSE error (defined in (9)), training time, number of target captures and number of collisions, considering 5 robot runs each of 20,000 timesteps and with a different randomly generated reservoir $\mathbf{W}_\mathrm{r}^\mathrm{r}$. The training time includes the time to generate the matrix $\mathbf{X}$ and to compute (8) using an Intel Core2 Duo processor-based system. During a robot run, there are three switching events as in Fig. 10. The last column of the table presents the percentage of successful runs which have resulted in correctly performing the selected behaviors for all three events of behavior switching during the respective simulation. It can be observed that as the reservoir has more units, the performance of the resulting RC-based controller increases, e.g., by decreasing the number of collisions, although the training time also increases. For reservoirs containing more than 400 neurons, the resulting RC-based controllers are always stable, i.e., the selected task (EE or TS) is performed reliably. With a proper initialization of the reservoir weights, even small reservoirs with 100 units can perform these navigation tasks very well. As this small reservoir must be randomly generated, this *proper* initialization is obtained by generating reservoirs and testing the resulting

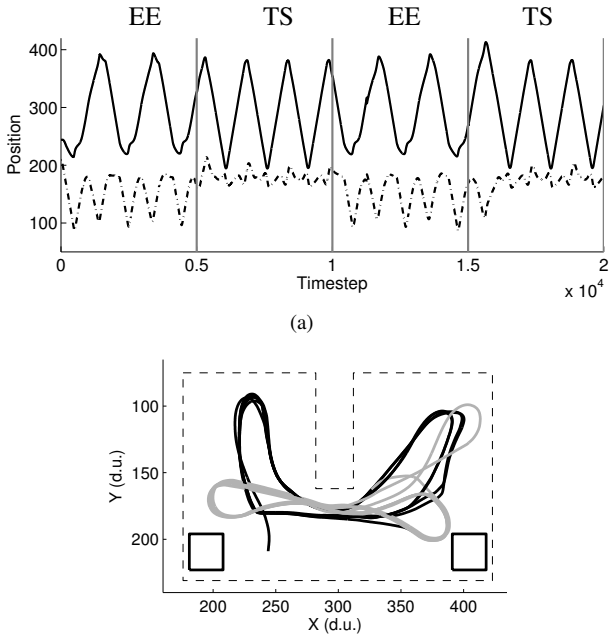


Fig. 10. Results for environment S1. (a) The coordinates of the robot are shown for 20,000 timesteps during the test phase. The solid and dashed lines are the x and y coordinates, respectively. Vertical gray lines represent the moments of behavior switching. (b) The corresponding trajectory of the robot in the Cartesian map. The solid black (gray) line represents the timesteps in which the selected behavior is the EE (TS) behavior.

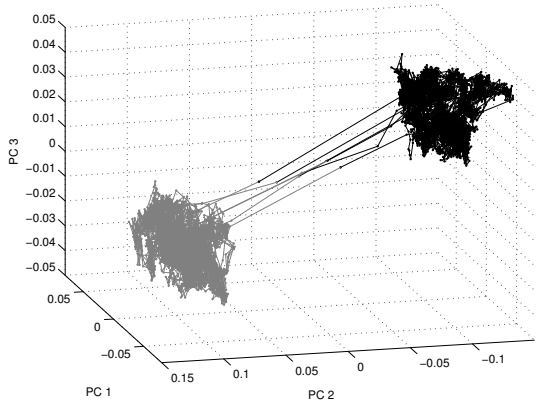


Fig. 11. Three principal components of the reduced dynamical system state space after applying PCA on the reservoir states during testing with the RC-based controller in environment S1. Gray and black lines represent trajectories associated with different selected behaviors. The input channel for behavior selection effectively shifts the operating point of the reservoir state space into different linearly separable sub-space *attractors*. There are six switching events, represented by the lines connecting both sub-space attractors. This figure is analogous to the fictitious example of Fig. 5.

TABLE II
MEAN RESULTS FOR DIFFERENT SIZE OF RESERVOIRS - ENVIRONMENT S1

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 %

controller until one solves the required task[6].

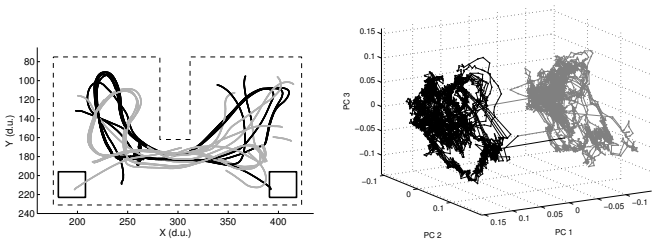
For testing the robustness of the RC controller to perturbations, a new experiment is accomplished in which the robot is artificially pushed in real time for several timesteps, and at least ten robot kidnappings take place during a simulation in S1 made of 20,000 timesteps and three switching events. Fig. 12 shows the trajectories for each behavior both in the environment space and reservoir state space. The left plot shows several displacements in the robot trajectory corresponding to events of robot kidnapping: after the robot is displaced, the controller tends to drive it back to the original attractor associated with a reactive behavior. The corresponding trajectories in the reservoir space (right plot) also shows that the property of linear separation existing between behaviors is maintained.

In order to test the generalization capabilities of the RC-based controller, a new dynamic environment S2 is considered which is different from the training environment (S1). The new environment (Fig. 8) is larger than S1, and has two targets, one located in the lower-left of the environment and another in the upper-right of the environment; it also contains two dynamic obstacles which have random *blinking* time periods, causing many disturbances in the robot behavior and perception. The simulation consists of 90,000 timesteps and 29 switching events, during which the robot captures the targets 16 times and collides 33 times against obstacles (mainly due to the sudden appearance of obstacles and unseen maneuvers). It has been observed that whenever a target was in the field of vision of the robot and the TS behavior was selected, the robot would seek and capture that target. The result in Fig. 13 shows the principal components of the reservoir states during this long simulation. The first 2 dimensions are correlated to the behavior selected by the external input whereas the third component encodes spatial information common to both EE and TS behaviors. Despite the many changes in environment configuration and stochasticity, this figure confirms that the learning of the RC network effectively embeds different robot behaviors into unseen dynamic environments.

For further comparison, we implemented a Multi-Layer Perceptron (MLP) with time-windowed inputs as a controller for the same task in S1. Resilient backpropagation was used to train the MLP[7], since it has given best results for the robot control task. Experiments were made with a time window of size $t_w = 2, t_w = 3$, generating a input layer of size 70 and 105, respectively, and with a hidden layer containing from

[6]On average, smaller randomly generated reservoirs have a lower probability of achieving a good performance and stable behavior than large reservoirs.

[7]A validation set was used to stop training.



(a) (b)

Fig. 12. Results for perturbations during navigation in environment S1. (a) Both EE and TS behaviors, given by black and gray trajectories, respectively, are perturbed several times by: robot kidnapping (at least 10 times), and one robot pushing (holding it over an small area by force for several iterations). (b) The corresponding principal components of the reservoir state space after applying PCA on the reservoir states. There are three switching events.

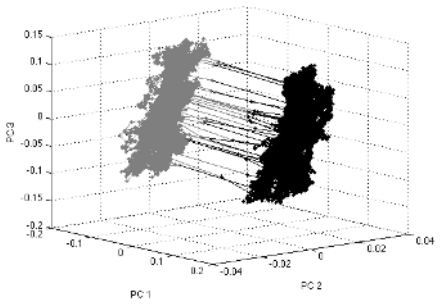


Fig. 13. Results for dynamic environment S2 using the controller trained in S1. The figure shows a trajectory formed by three Principal Components (PC) of the reservoir states over time. There are 29 behavior switch events during a simulation with 90,000 timesteps. While the first two PCs encode information on the specific behavior selected by the external input, the third PC encode spatial information, probably associated to obstacle avoidance skills.

6 up to 9 units. The average training time was 150 seconds. For each configuration of these two parameters, approximately $2/5$ of the experiments were close to the correct behavior. The main problems with this architecture was: very difficult to achieve the separation between EE and TS behaviors (e.g., eventually when executing the EE behavior, the robot would mistakenly capture a target); not enough generalization for collision avoidance (with an average of 36 collisions per simulation). The iterative nature of the training method seems to be one of the causes for the aforementioned problems, because it is not guaranteed to find the global minimum. For instance, only 1 out of 20 experiments generated a controller with perfect behavior (12 captures and zero collisions). Thus, the RC approach considerably benefits from a stable and reliable training method and a dynamic nature which allows for the relatively easy linear separation of robust behaviors in the dynamic reservoir space.

V. Second approach: Reinforcement learning of navigation behaviors

In the previous section, navigation behaviors have been learned in a supervised way with an one-shot learning process which uses examples consisting of sequences of the desired sensory-motor coupling.

In this section, instead, RC networks are used to approximate the state-action value function ($Q(s,a)$) in non-Markovian reinforcement learning navigation tasks, where the environment is partially observable (as in [42]). Under this scheme, an alternating sequence of policy improvement (samples generation from environment interaction) and policy evaluation (network training) steps are performed, the system is able to iteratively shape navigation attractors so that, after convergence, the robot can perform a well-formed behavior towards the goal.

A. Reservoir Computing for Q-value Approximation

In fitted Q iteration [43], samples in form of tuples

$$(s_t, a_t, r_t, s_{t+1}), \quad t = 1, \cdots, I,$$

are generated from interaction with the environment and collected in a training dataset. Training the system is done offline using the collected samples under a supervised learning framework: usually, a regression algorithm is used to learn the state-action value function, by defining the input and the desired output as follows:

$$\mathbf{u}[t] = (s_t, a_t), \tag{10}$$

$$\hat{y}[t] = r_t + \gamma \max_a \hat{Q}_{N-1}(s_{t+1}, a), \tag{11}$$

where: s_t, a_t and r_t are the state, action and reward at time t, respectively; N is the iteration of the training process; and γ is the discount factor. Using the dataset of input-output pairs $(\mathbf{u}[t], \hat{y}[t])$, the function $\hat{Q}_N(s,a)$ is induced with a regression algorithm.

In this section, an analog sigmoidal RC network or Echo State Network (ESN) is used to model the critic, that is, the Q-value [44] function, in non-Markovian environments. Given a partially observable state vector $\tilde{\mathbf{s}}$ and an action a as input, the goal is to approximate the expected future sum of rewards, the Q-value for the pair $(\tilde{\mathbf{s}}, a)$, using an RC network as approximation method. The randomly generated reservoir can convert non-Markovian state-spaces into Markovian state-spaces due to its characteristic fading memory of previous inputs. This method is similar to fitted Q iteration [43], [45] and least squares policy iteration [46] in that it is based on batch offline training and approximates the value function in an iterative way.

In [42], the RC network is used in reinforcement learning control tasks such as the mountain car problem and the more complex acrobot swing-up task. The input to the reservoir is a vector $\mathbf{u}[t]$ composed of a partially observable state $\tilde{\mathbf{s}}$, such as the position of the car or the joint angles of the acrobot (so, excluding the velocity component), and an action a, and the only output is trained to approximate the state-action value function.

As $\hat{Q}(\tilde{s}, a)$, the desired output $\hat{y}$, can be approximated by a sum of future rewards over a finite time horizon h [42], equations (10) and (11) can be rewritten, in the case of a non-Markovian environment:

$$\mathbf{u}[t] = (\tilde{s}_t, a_t), \tag{12}$$

$$\hat{y}[t] \approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^h r_{t+h} \tag{13}$$

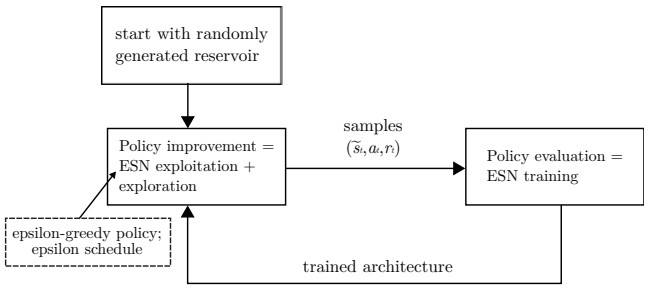


Fig. 14. Approximate Policy Iteration: Policy improvement + Policy evaluation. The iterative policy learning consists of: generation of samples by interacting with the environment using a ε-greedy policy and the trained architecture (policy improvement); and of training the architecture (in this case, the RC network) to approximate the state-action value function with a regression algorithm using the dataset generated during policy improvement. $\tilde{s}$ is a partially observable state, characterizing a non-Markovian task which should be handled by the RC network.

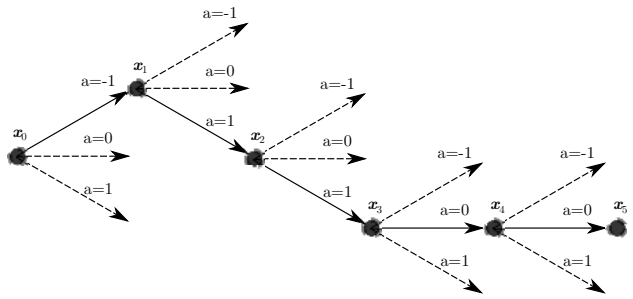


Fig. 15. Evolution of the reservoir state $\mathbf{x}[t]$ over time as the operator arg max is applied to the RC network. Dashed lines represent reservoir states which generated suboptimal paths during the application of arg max operator. The real path followed by the reservoir is given by solid lines.

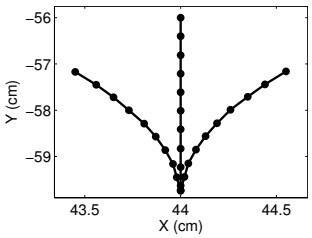


Fig. 16. Motor primitives or basic behaviors: left, forward and right.

The training is accomplished in an iterative way and consists of a sequence of policy improvement and policy evaluation steps (see Fig. 14). During policy improvement, new samples $(s_t, a_t, r_t), t = 1, ..., I$ are generated using a ε-greedy policy and the trained architecture. I is the number of samples generated during one iteration of the policy improvement stage, which is set to $I = 1000$. During policy evaluation, the training input-output pairs $(\mathbf{u}[t], \hat{y}[t]), t = 1, ..., E$ are generated using (12) and (13), respectively, and the RC network is trained on a subset of the dataset generated through interaction with the environment. This subset corresponds to a sliding window of samples of size E, such that only the most recent $E = 40,000$ samples are used for training. During the iterative policy learning process, the ε-greedy policy follows a learning schedule where the exploration is intense at the beginning of the process and monotonically decreases towards the end of the experiment. This is accomplished by varying ε according to a predefined schedule [42] (given in Section V-D).

The equations of the model and its training method, linear regression, are described in Section III-A. The equation for the readout output $y[t]$, which models the state-action value function in this section, is given by (6).

The **exploitation of the RC network** for the **control task** is based on the following equations:

$$a_{opt}[t+1] = \arg\max_a \left(y[t+1]\right) \tag{14}$$

$$a_{opt}[t+1] = \arg\max_a \left(\begin{bmatrix} \mathbf{W}_{\mathrm{r}}^{\mathrm{o}} & \mathbf{W}_{\mathrm{i}}^{\mathrm{o}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_a[t+1] \\ \tilde{\mathbf{s}}[t] \\ a \end{bmatrix} \right), \tag{15}$$

where $\mathbf{x}_a[t+1]$ is a internal reservoir state which is *dependent on the action a tested during the application of* arg max:

$$\mathbf{x}_a[t+1] = f\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}} \mathbf{x}[t] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}} \begin{bmatrix} \tilde{\mathbf{s}}[t] \\ a \end{bmatrix} \right).$$

This means that the reservoir state is *freezed* at timestep t, and to choose the optimal action, the arg max function runs the reservoir for each value of action a always starting at the same reservoir state $\mathbf{x}[t]$ from timestep t. For instance, Fig. 15 shows how the reservoir state evolves over time by using the argmax function on three possible values for action a $(-1, 0, 1)$.

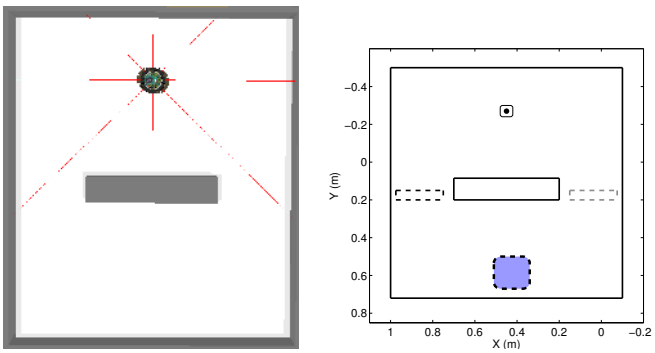
B. Motor Primitives

There are three motor primitives or basic behaviors in the low-level control module, which steer the 2 stepper motors of the e-puck robot: forward (left wheel: 500 steps/s; right wheel: 500 steps/s), left (left wheel: 250 steps/s; right wheel: 500 steps/s), and right (left wheel: 500 steps/s; right wheel: 250 steps/s). These motor primitives are executed for a period of 11 timesteps in the simulator (704 *ms*). See Fig. 16 for a graphical representation of the trajectories given by each of the motor primitives. It is relevant to observe that each primitive is inherently stochastic once the robot wheels can not reproduce the same trajectory due to non-systematic noise originated from wheel-slippage or irregularities of the floor. The motor primitives are designed to simplify the control task, by reducing the action space to 3 discrete actions.

C. Experiments

The robot task is to learn context-dependent navigation attractors in a partially observable environment. The environment is a rectangular arena with an obstacle between the robot and the goal location, as it can be seen in Fig. 17(a). During a simulation experiment, each episode starts with the robot located in the upper part of the room with position randomly chosen from a small interval defined by the solid rectangle in Fig. 17(b); the initial orientation of the robot is South, with small uniform noise added in the range $[0, 1.2]$ degrees. The robot is controlled according to a ε-greedy policy. The architecture is trained using the scheme depicted in Fig. 14 and explained in Sections V-A.

The task of the robot in this environment consists of navigating to the goal location, given by the light blue dashed box in Fig. 17(b), through the left or right part of the environment,



(a) (b)

Fig. 17. Rectangular environment with an obstacle between the robot and the goal location. (a) 3D environment in Webots, with the e-puck robot in the upper part. (b) Representative map of the environment in two dimensions. The box with a point inside represents the possible starting positions for the robot (randomly chosen), while the black and gray dashed rectangles represent the possible circumvention areas (dependent on the initial transient stimulus) which the robot has to use to reach the goal, represented by dashed box in light blue color.

shown by black and gray dashed rectangles in the same figure, depending on a previously received stimulus from the environment. This temporary stimulus can be implemented through the presence/absence of an object in the environment, the on/off of a light source, or the existence/absence of a sound. In the current experiments, this is simply implemented as an additional input signal to the reservoir which is 1.5 whenever the trajectory towards the goal should be done via the left side and -1.5 when the this trajectory should be performed via the right side. This extra signal is present for $2.1s$ in the beginning of each episode, during which the robot is not able to go left or right but only slowly forward (meant not to bias learning). After the initial period of $2.1s$, this extra input becomes zero.

One episode is finished whenever the robot reaches the goal performing the correct trajectory, hits against an obstacle, or when the length of the episode is greater than 60 timesteps. The reward r_t is always -1, unless the robot is at the goal location, when $r_t = 0$. When an episode ends, the input and desired output can be computed according to equations (12) and (13).

D. Settings

Table III shows the parameter configuration for the RC network, with critical parameters in bold. The inputs $\mathbf{u}$ to the network are 8 frontal distance sensors, scaled to the interval [0,1], an action $a \in \{-1,0,1\}$ and an additional input for the temporary stimulus.

The ε parameter for the policy, which corresponds to the probability of selecting random actions at each timestep, is selected from an arbitrarily chosen vector $[0.9,0.8,0.6,0.5,0.4,0.3,0.1,0.01]$, similarly to [42]. The particular timesteps in which ε changes follows a learning schedule chosen as [40, 140, 190, 220, 240, 260, 310, 330]$*10^3$ timesteps. This means, for instance, that during the first 40,000 timesteps, $\varepsilon = 0.9$. The finite time horizon in (13) is $h = 40$.

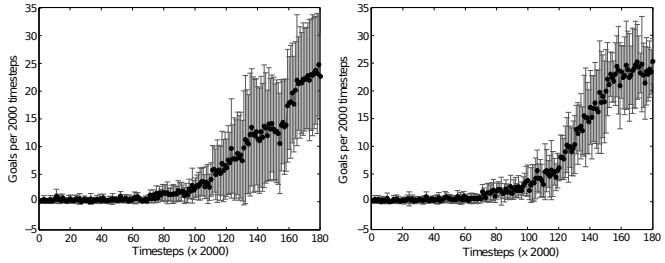
TABLE III
PARAMETER CONFIGURATION FOR RC NETWORK

Number of input channels	$n_i = 10$
Input connection fraction	$c_i^r = 0.5$
Input scaling	$\nu_i^r = 0.14$
Input downsampling	$d_t = 1$
Input to output connections	yes
Bias connection fraction	$c_b^r = 1$
Bias scaling	$\nu_b^r = 0.2$
Reservoir size	$n_r = 400$
Reservoir connection fraction	$c_r^r = 0.1$
Spectral radius	$\rho(\mathbf{W}_r^r) = 0.9$
Leak rate	$\alpha = 0.1$
Number of output channels	$n_o = 1$
Output feedback to reservoir	no

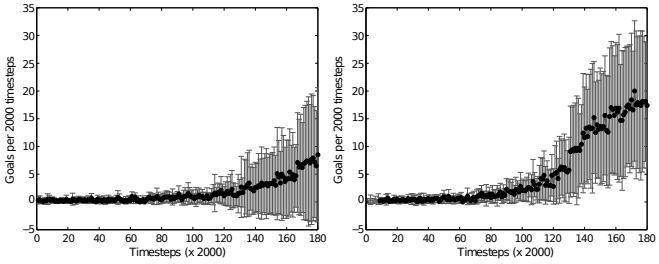
The discount factor is $\gamma = 1$, which defines a shortest-path problem.

The regression learning procedure for the reservoir architecture is executed every 1,500 timesteps considering the last $E = 50,000$ generated samples as learning window. These samples used for learning are generated from the interaction of the reservoir with the environment, while samples resulting from random actions are not taken into account during learning.

E. Results



(a) ESN



(b) The same network but without recurrent connections

Fig. 18. Average number of goals achieved per two thousand timesteps for 10 simulation experiments. The graphs at the left side represent the goals achieved via the *left trajectory*, while the graphs at the right represent the goals achieved via *right trajectory*. Error bars represent the standard deviation between runs. (a) Using the reservoir architecture presented in this section. (b) Using the same architecture, but without internal memory by setting $\mathbf{W}_{\text{res}}^{\text{res}} = 0$.

In order to evaluate the proposed robot navigation task using the ESN, the mean number of goals achieved per 2×10^3 timesteps considering *left and right trajectories* separately is

shown in Fig. 18(a). As time evolves, exploration decreases and the number of goals achieved via left and right trajectories (represented by black and blue lines, respectively) increases, which shows the capability of the architecture to learn short-term temporal dependencies in robot navigation tasks.

In Fig. 18(b), the mean number of achieved goals is computed using a memoryless architecture, implemented by simply setting the reservoir weights $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ to zero. It is possible to observe that the system does not learn the task correctly, preferring the *right trajectory* over the *left trajectory* in most of the experiments because the number of goals increases for the right navigation attractor (in blue) and decreases for the left attractor. Thus, without the fading memory of the reservoir, it is not possible to learn these navigation attractors correctly, because a memoryless architecture does not hold the temporary stimulus for future moments.

A single ESN can model multiple navigation attractors in a reinforcement learning task. These attractors, in the context of reinforcement learning, are dynamic, because the agent-environment interaction changes over time. Fig. 19(a) shows how these dynamic attractors evolves during the learning process. In the beginning, the two navigation attractors are not well formed, also because exploration is very high. In that stage, the system performs several possible trajectories due to random actions. As the simulation advances, the dynamic attractors are shaped so that the robot reaches the goal location performing a trajectory which is dependent on the initial temporary stimulus given at the beginning of the run.

Fig. 19(b) shows the principal components resulting from applying PCA on the reservoir states for the last episodes of simulation of Fig. 19(a). The principal component 3 encodes information used to follow the correct trajectory at the left or right side, thus forming a short-term memory responsible for holding the initial temporary stimulus. Fig. 19(c) shows that, after convergence of the learning process, the principal components form different trajectories in the state space according to the past stimulus given at the beginning of the episode.

One might use evolutionary methods to train RNNs in reinforcement learning tasks [47], [48], [49], but since the training of RC networks is not a problem as usually would be for traditional RNNs (because the recurrent reservoir is left untrained), the use of RC networks under a policy iteration scheme as shown above seems particularly fit to learning non-Markovian tasks.

VI. Third approach: Hierarchical architecture for goal-directed navigation

So far, RC networks have been used to generate behaviors under two different learning paradigms: supervised learning and reinforcement learning. Both approaches learn navigation attractors, either in an one-shot learning process (with examples given by a supervisor) or iteratively through interaction with the environment. Besides, the different behaviors have been formed and discriminated in the dynamic reservoir space by shifting the operating point of the reservoir with an external binary input channel.

Now, in this section, the transition from one behavior to another one is not done via an external input channel as before, but, instead, is accomplished through trained hidden units. These hidden units are responsible for autonomously detecting contextual switches, indicating, for example, when a robot is crossing a specific boundary from one room to another one in the environment. In this way, the change to a new behavior can be made dependent on the internally predicted context.

This ultimately leads to a system which can generate particular sequences of basic behaviors in an autonomous way for reaching a specific goal in a multi-room environment. In practical terms, this section presents an hierarchical architecture, composed of two modules: a localization module and a navigation module which operate at slow and fast timescales, respectively. The former module is trained to predict the current and the previously visited room based on the current distance sensors' readings, whereas the latter is trained to steer the robot in a goal-directed manner based on the input signals received from the localization module, distance sensors, and the target room. After training this multiple timescale hierarchical architecture with examples of navigation routes in simulated environments, the resulting RC-based controller is able to successfully navigate to specific target rooms in both simple and large unknown environments composed of many rooms.

A. Learning to Navigate to Goals by Imitation

The imitation learning procedure consists of two stages as follows.

1) Data Generation and Collection. In this stage, several examples of routes through the environment are generated, in which the robot navigates from a starting room to a destination room according to a predefined algorithm which uses *primitive reactive behaviors* to steer the robot in different modes. All required data for training are collected during this stage such as: distance sensors and destination room (which will be used as input channels); and the currently and previously visited robot rooms and desired motor actuators (for desired hidden or output units).
2) Training. The second stage involves the training of the RC networks with the data generated in the first stage. Afterwards, the trained RC-based navigation system can be used to drive the robot to specific destination rooms given as input.

To actually generate examples of navigation routes, two primitive reactive behaviors or navigation attractors are used to steer the robot through different paths inside a room. They are called *Left attractor* and *Right attractor*. Fig. 20 shows how these primitive behaviors can be used in sequence to generate complete paths to a destination room in an hypothetical environment. As a matter of simplicity, both primitive behaviors are implemented by different Braitenberg vehicles [50], whose motors' outputs consists of a linear combination of the current sensory readings (i.e., a linear sensory-motor mapping). The Braitenberg vehicle which avoids obstacles more intensely at the left side than at the right side forms a reactive **Left navigation attractor**. The **Right navigation attractor** is constructed in a similar way. These primitive

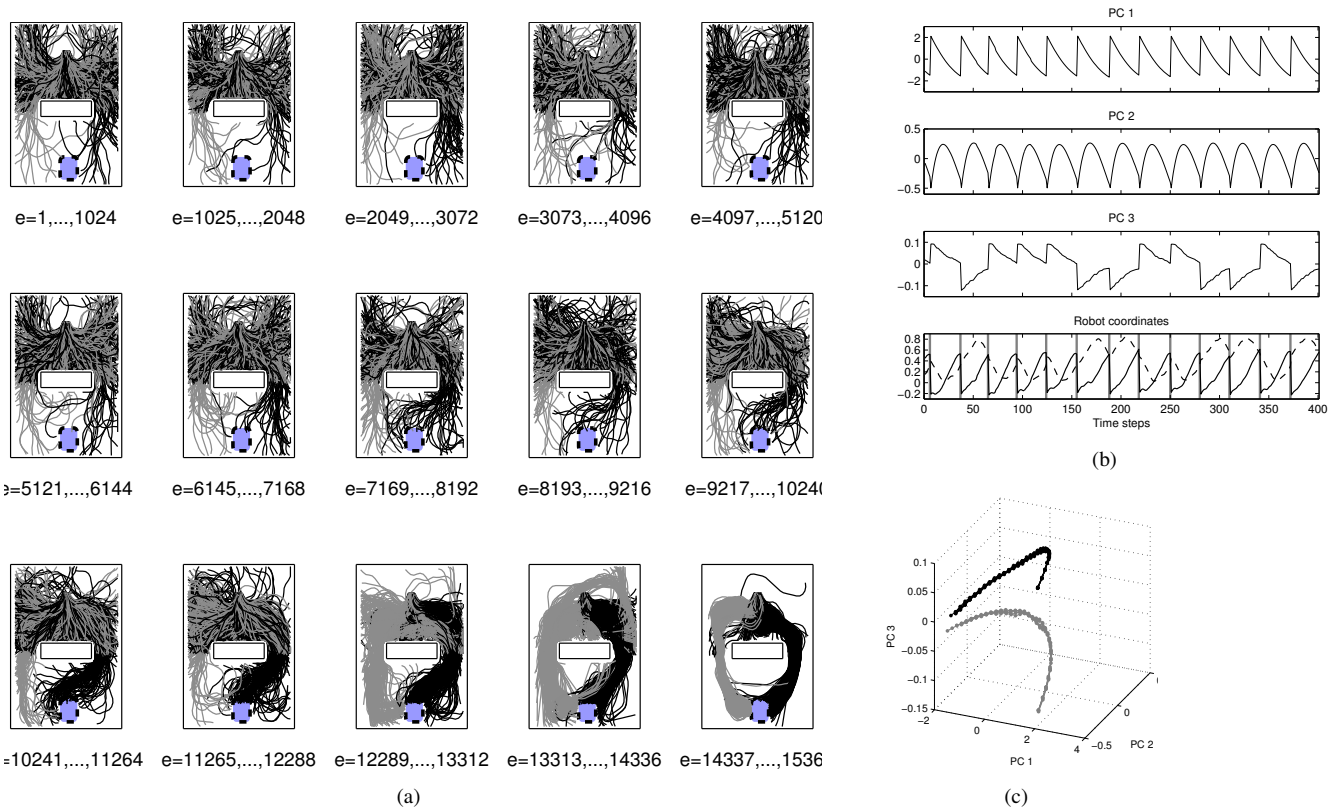


Fig. 19. (a) A sequence of robot trajectories as learning evolves, using the ESN. Each plot shows robot trajectories in the environment for several episodes during the learning process. In the beginning, exploration is high and several locations are visited by the robot. As the simulation develops, two navigation attractors are formed to the left and to the right so that the agent receives maximal reward. (b) Three principal components (PC) over time after applying PCA on the reservoir states, at the end of the simulation corresponding to last episodes in Fig. 19(a). The fourth plot shows the robot coordinates x, y over time in the environment. The gray vertical lines delimit different episodes. These plots were made disregarding the initial timesteps where the temporary stimulus is given, i.e., those initial timesteps were removed. The PC 3 encodes information used to follow the correct trajectory (left or right), thus forming a short-term memory responsible for holding the initial stimulus. (c) *Sub-space attractors* in the reduced dynamical system space for *left and right* navigation trajectories. The plot shows a 3D state space of the principal components, where gray and black lines represent different (left and right) trajectories in the environment, which are dependent on the previously received transient external stimulus.

behaviors form **spatial attractors** since they tend to follow *cyclic sensory-motor patterns in space* in static environments.

In the dynamical system space of the reservoir, *sub-space attractors* are formed resulting from the sensory-motor coupling which is learned with data collected using the two primitive behaviors. In other words, the reservoir should learn to reproduce the same context-dependent sensory-motor coupling, where each context transition (entering a room through a specific door) causes a change in the sensory-motor coupling (or navigation attractor). As the reservoir-based navigation system is tightly coupled with the environment, spatial navigation attractors once projected into the dynamical system space can be seen as sub-space attractors shifted by internal and/or external context switches. Fig. 21 shows the corresponding left and right sub-space attractors in a simplified bi-dimensional dynamical system space for the sequence of spatial navigation attractors shown in Fig. 20. Starting at room 1, the robot gets an external input for the goal destination, indicated by the transition given by the dashed arrows, and performs a series of primitive behaviors which are fired by internal transitions, represented by solid arrows, which ultimately lead to the final destination. For instance, the transition r.2 g.5 signals that the robot entered room 2 from room 1 while its destination (goal) is room 5. These internal transitions will be modeled by a *localization* reservoir, which predicts the current and previously visited room. The *navigation* reservoir models the sensory-motor coupling given by navigation attractors, whose operation is modified by the guidance of the localization reservoir. These two RC networks form an hierarchical architecture described in the following section.

B. Hierarchical RC Architecture

The Hierarchical Reservoir Computing (HRC) controller is composed of two RC networks or modules: the localization and the navigation modules (see Fig. 22). It is relevant to observe that the localization reservoir operates at a much slower timescale than the navigation reservoir since transitions between rooms are very sporadical, requiring a reservoir with slow dynamics (achieved by using a low leak rate α) when compared to the required quick reaction of reservoirs implementing navigation behaviors.

The learning process is divided in two stages:

1) 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.

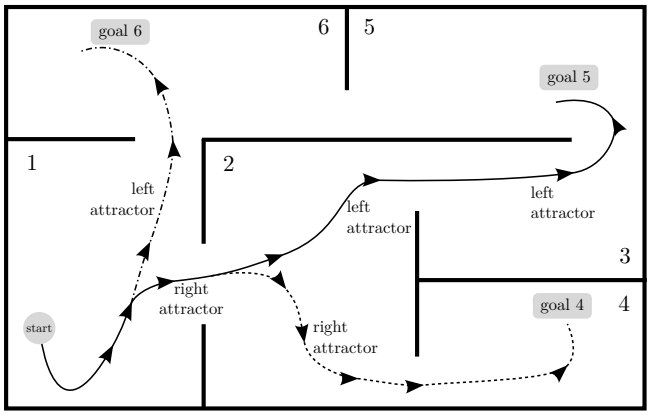


Fig. 20. Example of goal-directed navigation as a sequence of reactive *navigation attractors* or behaviors: *left attractor* and *right attractor*. The plot shows an hypothetical environment with 6 rooms and robot trajectories represented by solid and dashed lines, with arrows indicating the orientation of the robot. The two simple reactive behaviors, i.e., left and right attractors, lead the robot to different paths in a room. Three different trajectories leading to goals 4, 5 and 6 are shown in the environment. For instance, the mobile robot reaches goal 5, starting at room 1 and choosing: right attractor, left attractor and left attractor. Examples of routes like these are generated for the imitation learning process.

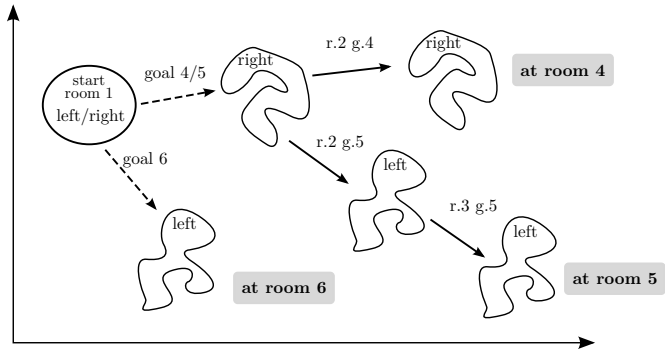


Fig. 21. Simplistic view of navigation attractors in bi-dimensional dynamical system space corresponding to the routes to goals 4, 5 and 6 shown in Fig. 20. The circle represents the starting position of the robot, which can be in left or right attractor. Dashed lines represent transitions between sub-space attractors in the dynamical system space given by external input channels, while solid lines indicate transitions given by internal hidden activity, resulting from the internal predictions of the current and possibly the previously visited location, for instance (the transition r.2 g.4 is an abbreviation of *room 2* and *goal 4*, i.e., the robot is located at intermediate room 2, with room 4 as final destination). The goal rooms are reached after a sequence of sub-space attractors, representing simple reactive behaviors, has been performed.

2) Then, the **navigation module** is trained 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:

$$\begin{aligned}\mathbf{x}_{\text{loc}}[n+1] = \quad & (1-\alpha_{\text{loc}})\mathbf{x}_{\text{loc}}[n] + \alpha_{\text{loc}} f((\mathbf{W}^{\text{r}}_{\text{i\_loc}}\mathbf{u}_{\text{dist}}[n] + \\ & \mathbf{W}^{\text{r}}_{\text{r\_loc}}\mathbf{x}[n] + \mathbf{W}^{\text{r}}_{\text{b\_loc}})),\end{aligned} \tag{16}$$

$$\mathbf{y}_{\text{c}}[n+1] = \quad g(\mathbf{W}^{\text{out}}_{\text{c}}\mathbf{x}_{\text{loc}}[n+1]), \tag{17}$$

$$\mathbf{y}_{\text{p}}[n+1] = \quad g(\mathbf{W}^{\text{out}}_{\text{p}}\mathbf{x}_{\text{loc}}[n+1]), \tag{18}$$

where $\mathbf{y}_{\text{c}}$ and $\mathbf{y}_{\text{p}}$ 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 III-A1, but have new subscripts for identifying the localization reservoir.

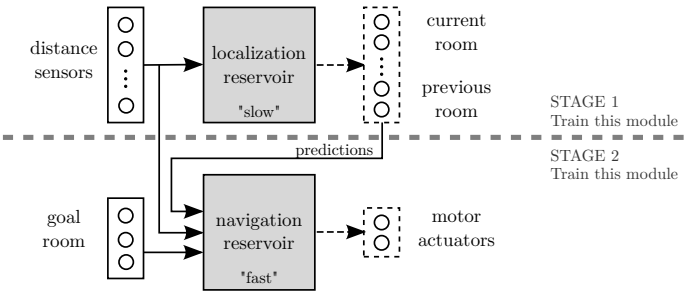


Fig. 22. Hierarchical architecture with localization and navigation modules. The navigation and localization reservoirs are randomly generated recurrent networks which are not trained, but left fixed. Trainable components (or weights) are shown in dashed lines. The sensory input feeds both reservoirs, being mapped to a high-dimensional space, where learning occurs. The navigation reservoir receives input also from the localization module and the target location and outputs the desired motor actuators. Stage 2 trains the navigation module using the **predictions** given by the localization module, trained in Stage 1.

Analogously, the equations for the navigation module are as follows:

$$\begin{aligned}\mathbf{x}_{\text{nav}}[n+1] = \quad & (1-\alpha_{\text{nav}})\mathbf{x}_{\text{nav}}[n] + \alpha_{\text{nav}} f((\mathbf{W}^{\text{r}}_{\text{i\_nav}}\mathbf{u}_{\text{multi}}[n] \\ & + \mathbf{W}^{\text{r}}_{\text{r\_nav}}\mathbf{x}[n] + \mathbf{W}^{\text{r}}_{\text{b\_nav}})),\end{aligned} \tag{19}$$

$$\mathbf{y}_{\text{nav}}[n+1] = \quad g(\mathbf{W}^{\text{out}}_{\text{nav}}\mathbf{x}_{\text{nav}}[n+1]), \tag{20}$$

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

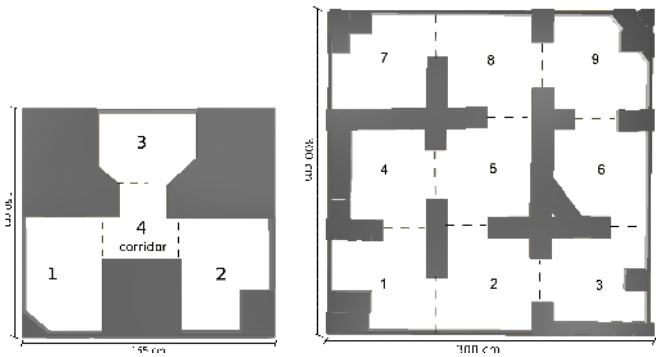
$$\mathbf{u}_{\text{multi}}(t) = [\mathbf{u}^T_{\text{dist}}(t)\mathbf{y}^T_{\text{c}}(t)\mathbf{y}^T_{\text{p}}(t)\mathbf{u}^T_{\text{goal}}(t)]^T.$$

The weight matrices $\mathbf{W}^{\text{out}}$ in Equations (17), (18) and (20) are trained using linear regression as explained in Section III-A2. All other weight matrices connecting to the reservoir are randomly generated at the beginning of the experiment and left fixed.

VII. Experiments

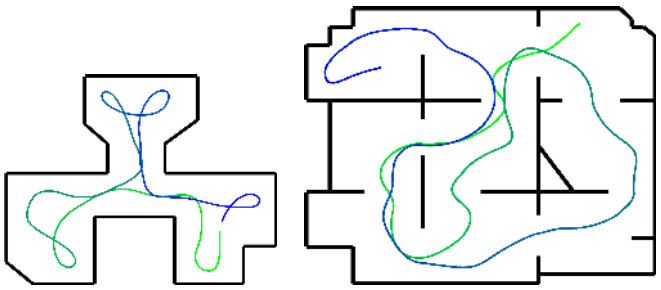
The proposed HRC architecture was evaluated in two environments. Environment E4 is composed of three rooms connected by a central corridor (see Fig. 23). A second, larger environment E5 is made of 9 rooms with open doors connecting them.

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 an initial room to a target room (see Fig. 24(a) for an example) - there are 6 possible routes in environment E4.



(a) E4 (b) E5

Fig. 23. Webots 3D 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 rooms are 1, 3, 7 or 9). Dashed lines represent boundary limits between rooms.



(a) E4 (b) E5

Fig. 24. Samples of robot trajectories used as training examples for the HRC controller. (a) Trajectory in E4. (b) Trajectory in E5.

TABLE IV
PARAMETER CONFIGURATION FOR ENVIRONMENT E4

Module	Localization	Navigation
Number of input channels	$n_i = 8$	$n_i = 19$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.5$
Input scaling	$\upsilon_i^r = 1$	$\upsilon_i^r = 1$
Input downsampling	$d_t = 10$	$d_t = 5$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 400$	$n_r = 400$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_r^r) = 0.98$	$\rho(\mathbf{W}_r^r) = 0.98$
Leak rate	$\alpha = 0.01$	$\alpha = 1$
Number of output channels	$n_o = 8$	$n_o = 2$
Output feedback to reservoir	no	no

TABLE V
PARAMETER CONFIGURATION FOR ENVIRONMENT E5

Module	Localization	Navigation
Number of input channels	$n_i = 8$	$n_i = 30$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.5$
Input scaling	$\upsilon_i^r = 1$	$\upsilon_i^r = 1$
Input downsampling	$d_t = 10$	$d_t = 5$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 400$	$n_r = 400$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_r^r) = 0.98$	$\rho(\mathbf{W}_r^r) = 0.98$
Leak rate	$\alpha = 0.01$	$\alpha = 1$
Number of output channels	$n_o = 18$	$n_o = 2$
Output feedback to reservoir	no	no

The second environment E5 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 environment E4. See Fig. 24(b) for an example of robot trajectories generated with the supervisor controller.

VIII. SETTINGS

For both environments E4 and E5, the two datasets of 500,000 and 100,000 samples were downsampled by a factor of $d_t^{\text{loc}} = 10$ and $d_t^{\text{nav}} = 5$ respectively (values empirically chosen to give best performance), resulting in two new datasets of 50,000 and 20,000 samples for training the localization and the navigation module, respectively. As the these sampling rates are different from each other, signals from the localization reservoir $\mathbf{y}_c$ and $\mathbf{y}_p$ are upsampled to the same sampling rate of the navigation reservoir before they are used as input to that module.

The parameter configuration is given in Table IV for environment E4 and Table V for environment E5. Some of these parameters are described in Section III-A. As it can be seen from these tables, the experiments on both environments use the same parameter configuration, except for the number of outputs n_o of the localization module, and the number of inputs n_i for the navigation reservoir. For environment E5, $n_o^{\text{loc}} = 18$ (9 units for previously visited room and 9 for the current room) and $n_i^{\text{nav}} = 30$ (18 from the localization module + 4 goal inputs + 8 distance sensors). The critical parameters α and d_t (shown in bold in the tables above) 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 by trial and error).

IX. RESULTS

The test data for environment E4 consists of 50,000 samples downsampled to 5,000 timesteps. The system 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 HRC system after training are shown in Fig. 25. The robot starts in one of the rooms in a 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 drawn such that its color incrementally changes from green to blue, representing the progress of the navigation. In Fig. 25(c), 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

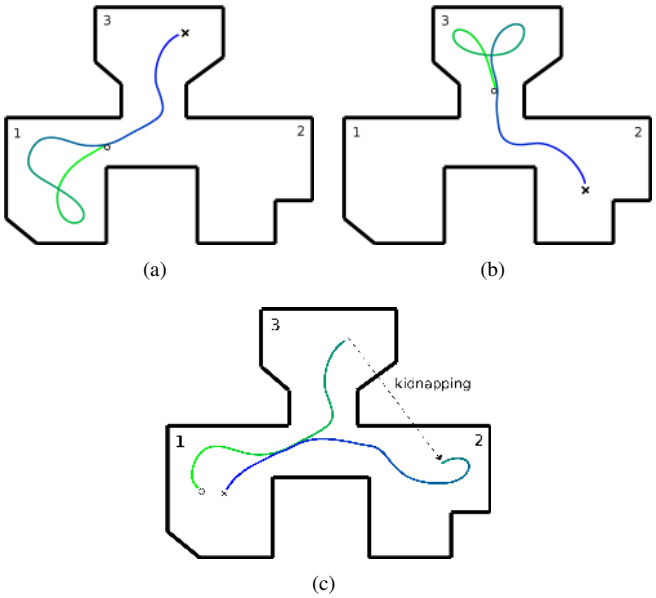
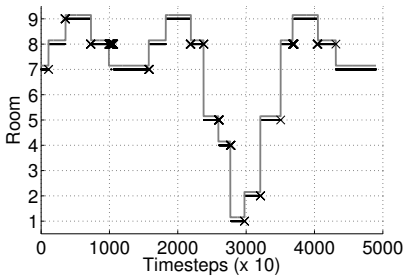
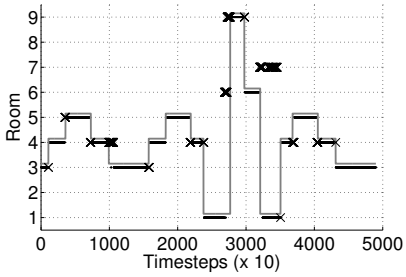


Fig. 25. Trajectories for robot driven by the HRC controller in environment E4. (a) Robot starts at room 1 and goes to room 3. (b) Robot starts at room 3 and goes to room 2. Starting and ending positions are marked with a circle and a cross, respectively. (c) The robot drives from room 1 to goal room 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.



(a) E5 - current room



(b) E5 - previous room

Fig. 26. Performance results of the localization module in environment E5. Predicted locations are represented by black points whereas solid grey lines are the true robot location. Black crosses represent mistakes.

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 was able to drive successfully to its destination (goal room 1), showing that it correctly recognizes the room the robot is located at, which in turn, affects the operation mode of the sensory-motor coupling of the navigation reservoir. This result is consistent across multiple trials and experiments. In 63 routes that were evaluated, the HRC controller has been able to successfully drive the robot to the destination room in all cases without any collision.

The localization performance on test data for environment E5 is shown in Fig. 26(a). The system can detect the current and previously visited room 96.33% and 93.63% of the time, respectively. An example of a successful trajectory in environment E5 is shown in Fig. 27(a). The robot, driven by the HRC controller, starts at room 7 and reaches room 1 successfully. In 15 out of 23 runs, the robot was able to 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 path (see Fig. 27(b) for an example). This also shows that the HRC controller is robust to noise and unpredictable situations since it is able to reach the destination even though the robot looses itself for a moment when it mistakenly enters a room outside the shortest path. A summary of the experimental results is given in Table VI.

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.

TABLE VI
PERFORMANCE RESULTS IN NUMBER OF TRAJECTORIES

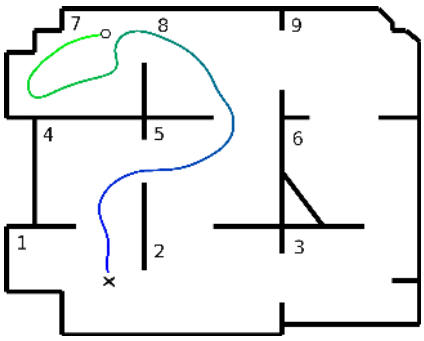
	Shortest Path	Task completion
Environment E4	63 out of 63 (100%)	100%
Environment E5	15 out of 23 (65%).	100%

Although navigation does not start in intermediate rooms in environment E5 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 the trained system can avoid dynamic unseen obstacles during testing while reaching the desired goal locations. This generalization capability is expected to work with the proposed architecture once it has been shown in Section IV that reservoir architectures can learn and generalize obstacle avoidance behaviors.

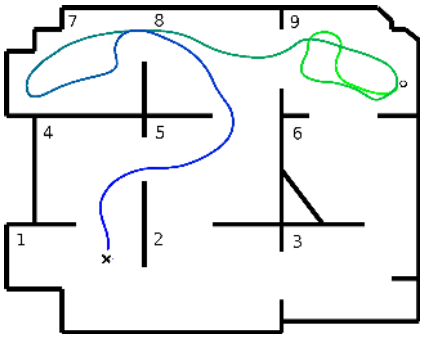
X. CONCLUSION

In this paper, three approaches have been presented on learning navigation behaviors for small mobile robots. It is assumed that these robots have only a few (from 8 to 17) noisy distance sensors for navigation, which could facilitate the application of these methods to commercial products in the field of service robotics.

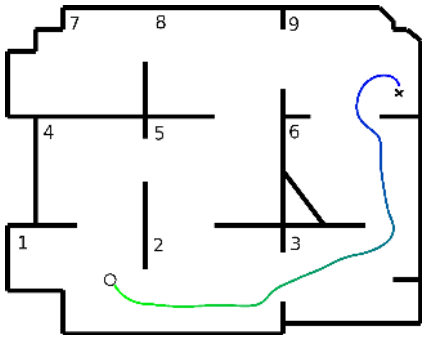
The common aspects for all approaches are mainly two-fold: they use Reservoir Computing networks for efficient recurrent neural network training, where the reservoir (an RNN) has fixed weights while only a readout output layer is trained; they are based on the concept of navigation attractors which correspond to reactive behaviors that can be embedded



(a) Start 7 - Destination 1



(b) Start 9 - Destination 1



(c) Start 1 - Destination 3 and Start 3 - Destination 9

Fig. 27. Trajectories for robot driven by the HRC controller in environment E5. Starting and ending positions are marked with a circle and a cross, respectively. (a) Starting at room 7 and going to target room 1 via rooms ($8 \rightarrow 5 \rightarrow 4$) (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). (c) Two routes: Starting at room 1 and going to target room 3 via room 2; and starting at room 3 and going to target room 9 via room 6.

into a dynamic system space (reservoir space) through robot-environment coupling after training.

The three approaches differ in two aspects: while the first and third approaches are based on a supervised learning framework for modeling directly the desired sensory-motor coupling, the second approach learns behaviors iteratively in a reinforcement learning way by modeling the state-action value function; whereas the first and second approaches model and discriminate behaviors by the use of an external binary input channel and a single RC network, the third approach makes use of an hierarchical structure in which hidden units predict contextual switches responsible for guiding the execution of reactive behaviors.

In summary, this work shows how an RC network can model increasingly complex behaviors with single and hierarchical networks, by either showing examples of behaviors or making use of rewards in a trial and error process. The proposed RC framework is based on the notion of *sub-space attractors*, which can be viewed as the projection of the reactive behaviors from the sensory space to the dynamic reservoir space. This projection enables the learning of multiple behaviors since the high-dimensional space of the reservoir makes possible their linear discrimination.

There are several research directions to be explored in the future. In the context of animal spatial navigation, the hierarchical architecture shown in this work could be used to generate future possible trajectories according to the selected behavior (as an extension to [51]), known as planning in the robotics literature, and as mental simulation in cognitive science [52], [53]. By examining all possible future routes (based on past experiences), the one that leads to a reward could then be chosen to be executed. A second research direction is to automate the segmentation of complex behaviors into a set of smaller and basic reactive behaviors (e.g., as motor primitives) which, in turn, could be sequenced to be executed in a hierarchical architecture such as the one presented in this work. However, instead of separating segmentation and learning architecture, they could be merged into an architecture which autonomously segment the complex behaviors into simpler ones as well as learns to switch from one behavior to the next one just by demonstration of the complex behavior. Similar works in literature which implement this type of segmentation are [54], [55]. Extensions for the second approach (Sec. V) include the generation of more complex behaviors such as those with more longer-term temporal dependencies and goal-directed navigation in larger room-based environments. Additionally, the supervised learning of spatial features in the hierarchical architecture could be replaced by an unsupervised method such as the one proposed in [56], such that no manual labeling of location data is necessary.

ACKNOWLEDGMENT

The authors gratefully acknowledge the contributions of Dries Van Puymbroeck and Stefan Depeweg to the experiments in this paper.

REFERENCES

[1] R. Brooks, "New approaches to robotics," *Science*, vol. 253, no. 5025, pp. 1227–1232, 1991.
[2] R. Arkin, *Behavior-Based Robotics*. Cambridge, MA: The MIT Press, 1998.
[3] M. Wilson, "Six views of embodied cognition," *Psychonomic Bulletin & Review*, vol. 9, no. 4, pp. 625–636, 2002.
[4] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[5] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.
[6] D. Buonomano and W. Maass, "State-dependent computations: Spatiotemporal processing in cortical networks," *Nature Reviews Neuroscience*, vol. 10, no. 2, pp. 113–125, 2009.
[7] 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.

[8] T. Yamazaki and S. Tanaka, "The cerebellum as a liquid state machine," *Neural Networks*, vol. 20, no. 3, pp. 290–297, 2007.

[9] H. Jaeger and H. Haas, "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication," *Science*, vol. 304, no. 5667, pp. 78–80, Apr. 2004.

[10] T. Waegeman, M. Hermans, and B. Schrauwen, "MACOP modular architecture with control primitives," *Frontiers in Computational Neuroscience*, vol. 7, no. 99, pp. 1–13, 2013.

[11] C. Emmerich, R. F. Reinhart, and J. J. Steil, "Multi-directional continuous association with input-driven neural dynamics," *Neurocomputing*, vol. 112, no. 18, pp. 47–57, 2013.

[12] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific Reports*, vol. 2, no. 287, pp. 1–6, 2012.

[13] F. Triefenbach, A. Jalalvand, K. Demuynck, and J.-P. Martens, "Acoustic modeling with hierarchical reservoirs," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 21, no. 11, pp. 2439–2450, Nov. 2013.

[14] M. Hermans and B. Schrauwen, "Recurrent kernel machines: computing with infinite echo state networks," *Neural Computation*, vol. 24, no. 1, pp. 104–133, 2012.

[15] E. A. Antonelo, B. Schrauwen, and D. 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*, Singapore, Oct. 2008, pp. 1843–1848.

[16] E. A. Antonelo, S. Depeweg, and B. Schrauwen, "Learning navigation attractors for mobile robots with reinforcement learning and reservoir computing," in *Proceedings of the X Brazilian Congress on Computational Intelligence (CBIC)*, Fortaleza, Brazil, Nov. 2011.

[17] E. A. Antonelo and B. Schrauwen, "Supervised learning of internal models for autonomous goal-oriented robot navigation using reservoir computing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, AK, May 2010, pp. 2959–2964.

[18] E. A. Antonelo, B. Schrauwen, and D. Stroobandt, "Event detection and localization for small mobile robots using reservoir computing," *Neural Networks*, vol. 21, no. 6, pp. 862–871, 2008.

[19] J. Tani, "Model-based learning for mobile robot navigation from the dynamical systems perspective," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 3, pp. 421–436, Jun. 1996.

[20] P. Verschure, B. Krose, and R. Pfeifer, "Distributed adaptive control: The self-organization of structured behavior," *Robotics and Autonomous Systems*, vol. 9, no. 3, pp. 181–196, 1992.

[21] P. Verschure, T. Voegtlin, and R. Douglas, "Environmentally mediated synergy between perception and behaviour in mobile robots," *Nature*, vol. 425, no. 6958, pp. 620–624, 2003.

[22] D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 3, pp. 396–407, Jun. 1996.

[23] 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*, vol. 15, no. 3, pp. 639–652, May 2004.

[24] M. Milford, *Robot Navigation from Nature: Simultaneous Localisation, Mapping, and Path Planning Based on Hippocampal Models*, ser. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2008, vol. 41.

[25] T. Stroesslin, D. Sheynikhovich, R. Chavarriaga, and W. Gerstner, "Robust self-localisation and navigation based on hippocampal place cells," *Neural Networks*, vol. 18, no. 9, pp. 1125–1140, 2005.

[26] R. Chavarriaga, T. Strsslin, D. Sheynikhovich, and W. Gerstner, "A computational model of parallel navigation systems in rodents," *Neuroinformatics*, vol. 3, no. 3, pp. 223–241, 2005.

[27] M. Franzius, H. Sprekeler, and L. Wiskott, "Slowness and sparseness lead to place, head-direction, and spatial-view cells," *PLoS Computational Biology*, vol. 3, no. 8, pp. 1605–1622, 2007.

[28] R. Wyss, P. Knig, and P. F. M. J. Verschure, "A model of the ventral visual system based on temporal stability and local memory," *PLoS Biol*, vol. 4, no. 5, 2006.

[29] D. Filliat and J.-A. Meyer, "Map-based navigation in mobile robots: I. a review of localization strategies," *Cognitive Systems Research*, vol. 4, no. 4, pp. 243–282, 2003.

[30] O. Trullier, S. I. Wiener, A. Berthoz, and J.-A. Meyer, "Biologically-based artificial navigation systems: Review and prospects," *Progress in Neurobiology*, vol. 51, no. 5, pp. 483–544, Apr. 1997.

[31] C. Hartland and N. Bredeche, "Using Echo State Networks for Robot Navigation Behavior Acquisition," in *Proc. of the IEEE Int. Conf. on Robotics and Biomimetics*, Sanya, China, Dec. 2007, pp. 201–206.

[32] M. Lukosevicius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI - Knstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[33] H. Jaeger, M. Lukosevicius, and D. Popovici, "Optimization and applications of echo state networks with leaky integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, Apr. 2007.

[34] B. Schrauwen, J. Defour, D. Verstraeten, and J. Van Campenhout, "The introduction of time-scales in reservoir computing, applied to isolated digits recognition," in *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN 2007)*, ser. LNCS. Springer Berlin Heidelberg, 2007, vol. 4668, pp. 471–479.

[35] B. Schrauwen, L. Busing, and R. Legenstein, "On Computational Power and the Order-Chaos Phase Transition in Reservoir Computing," in *Advances in Neural Information Processing Systems 21 (NIPS 2008)*. Curran Associates, Inc., 2009, pp. 1425–1432.

[36] Y. A. Kuznetsov, *Elements of Applied Bifurcation Theory*. Springer, 1998.

[37] 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.

[38] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, "Memory versus non-linearity in reservoirs," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain, Jul. 2010, pp. 1–8.

[39] 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, BC, Jul. 2006, pp. 498–505.

[40] F. Mondada, "E-puck education robot," Sep. 2007, http://www.e-puck.org/.

[41] H. Jaeger, "Short term memory in echo state networks," German National Research Center for Information Technology, Tech. Rep. GMD Report 152, Mar. 2002.

[42] K. Bush, "An echo state model of non-markovian reinforcement learning," Ph.D. dissertation, Colorado State University, Fort Collins, CO, 2008.

[43] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, Apr. 2005.

[44] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, Mar. 1998.

[45] M. Riedmiller, "Neural fitted Q iteration first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005*, ser. LNCS, vol. 3720. Springer, 2005, pp. 317–328.

[46] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[47] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[48] F. Gomez and R. Mikkulainen, "Incremental evolution of complex general behavior," *Adapt. Behav.*, vol. 5, no. 3-4, pp. 317–342, Jan. 1997.

[49] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*, ser. Studies in Fuzziness and Soft Computing. Springer Berlin Heidelberg, 2006, vol. 192, pp. 75–102.

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

[51] 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.

[52] B. E. Pfeiffer and D. J. Foster, "Hippocampal place-cell sequences depict future paths to remembered goals," *Nature*, vol. 497, pp. 74–79, May 2013.

[53] F. Chersi, F. Donnarumma, and G. Pezzulo, "Mental imagery in the navigation domain: a computational model of sensory-motor simulation mechanisms," *Adaptive Behavior*, vol. 21, no. 4, pp. 251–262, Aug. 2013.

[54] J. Tani and S. Nolfi, "Learning to perceive the world as articulated: An approach for hierarchical learning in sensory-motor systems," *Neural Networks*, vol. 12, no. 7-8, pp. 1131–1141, Oct. 1999.

[55] Y. Yamashita and J. Tani, "Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment," *PLoS Comput Biol*, vol. 4, no. 11, pp. 1–18, Nov. 2008.
[56] E. A. Antonelo and B. Schrauwen, "Learning slow features with reservoir computing for biologically-inspired robot localization," *Neural Networks*, vol. 25, no. 1, pp. 178–190, Jan. 2012.