X Congresso Brasileiro de Inteligência Computacional (CBIC'2011), 8 a 11 de Novembro de 2011, Fortaleza, Ceará

LEARNING NAVIGATION ATTRACTORS FOR MOBILE ROBOTS WITH REINFORCEMENT LEARNING AND RESERVOIR COMPUTING

Eric Aislan Antonelo
ELIS Department - Ghent University
eric.antonelo@elis.ugent.be

Stefan Depeweg
Institute of Cognitive Science - University of Osnabrueck
stdepewe@uni-osnabrueck.de

Benjamin Schrauwen
ELIS Department - Ghent University
benjamin.schrauwen@elis.ugent.be

Abstract – .
Autonomous robot navigation in partially observable environments is a complex task because the state of the environment can not be completely determined only by the current sensory readings of a robot. This work uses the recently introduced paradigm for training recurrent neural networks (RNNs), called reservoir computing (RC), to model multiple navigation attractors in partially observable environments. In RC, the RNN with randomly generated fixed weights, called reservoir, projects the input into a high-dimensional dynamic space. Only the readout output layer is trained using standard linear regression techniques, and in this work, is used to approximate the state-action value function. By using a policy iteration framework, where an alternating sequence of policy improvement (samples generation from environment interaction) and policy evaluation (network training) steps are performed, the system is able to shape navigation attractors so that, after convergence, the robot follows the correct trajectory towards the goal. The experiments are accomplished using an e-puck robot extended with 8 distance sensors in a rectangular environment with an obstacle between the robot and the target region. The task is to reach the goal through the correct side of the environment, which is indicated by a temporary stimulus previously observed at the beginning of the episode. We show that the reservoir-based system (with short-term memory) can model these navigation attractors, whereas a feedforward network without memory fails to do so.
Keywords – reservoir computing, reinforcement learning, robot navigation .

1. INTRODUCTION

Mobile robot navigation systems have been modeled under several different types of techniques. The traditional robotics approach is based on probabilistic methods for robot localization, which seeks to solve the Simultaneous Localization And Mapping (SLAM) problem [1]. Environment constraints, sensor uncertainty and odometry have to be explicitly modeled in these systems. On the other hand, navigation systems modeled with computational intelligence techniques, such as neural networks and evolutionary systems, offer an alternative to modeling unknown environments without the need to know a priori information of the task or environment and can inherently deal with noisy environments [2, 3].

In particular, navigation tasks frequently need to be solved under the assumption that the state of the environment is partially observable, where the robot does not know the complete state of the environment just by reading its current sensors, such as its position in the environment. This is usually referred in the literature as a partially observable Markov decision process (POMDPs). Thus, the navigation system requires additional sources of information for building the complete state of the environment, such as using wheel encoders to estimate the traveled distance and the current robot position. Instead of explicitly modeling odometry, the works in [4, 5] show that a recurrent neural network (RNN) can detect the location of the robot in a supervised or unsupervised way by using an emerging computing paradigm called Reservoir Computing (RC) [6].

In Reservoir Computing, a parameterized dynamical system is used to project the input signal into a high-dimensional non-linear state-space. This dynamical system should possess a fading memory and it is most commonly implemented as a randomly generated sigmoidal RNN with fixed weights, commonly known as Echo State Networks [7]. Other forms of reservoir computing include: photonic reservoir computing [8], reservoir computing with cellular neural/nonlinear networks [9] and liquid state machine for spiking neural networks [10]. The advantage in RC is that only the readout output layer is trained, usually by employing standard linear regression methods, whereas the recurrent reservoir weights are left fixed. The advantages of RNN training with RC over traditional methods such as Backpropagation-Through-Time are: training is fast and convergence to the global optimum is guaranteed.

Reinforcement Learning (RL) rules are based on a reward signal $r[t]$, which represents the outcome (i.e., success or failure) of a learning trial. In immediate reinforcement learning, the reward signal is given at every timestep and is usually computed as

the *distance to the goal*. On the other hand, for RL methods with delayed reward signals the outcome is given only at the end of a trial. Whereas in the former case hints are given in every moment, the latter case defines a very less informative reward function which can delay the learning process, although it is more biologically plausible. RL algorithms such as *Q-learning* [**?**] learns an action value function $Q(s, a)$ which indicates the utility of an action a in a state s. It tries to maximize the mean expected future reward by using a *value iteration update* rule. Whereas these rules are applied in an online fashion, in fitted Q iteration [14] the $Q(s, a)$ is learned in batch mode with supervised learning techniques such as linear regression and artificial neural networks.

RC-based systems are usually trained in a supervised way. Some works in the literature use RC in a Reinforcement Learning (RL) framework to model partially observable environments [11, 12]. In [12], an ESN with multiple outputs in the readout layer, each one corresponding to the value of a discrete action, are trained online via a SARSA update [13] rule, whereas in [11], an ESN with one output representing the value function for a state-action input pair is trained in an iterative batch-mode way. Both works consider partially observable environments, but while [12] applies to discrete-world tasks, [11] consider more complex control tasks such as the acrobot swing-up task [13].

Function approximators are typically used to model the state-action value function in complex problems where the state space is continuous, and in most cases a ϵ-greedy policy on the value function is used for selecting the optimal action, where ϵ defines the probability of selecting random actions for exploration of the state space. In particular, sample-based methods like fitted Q iteration [14] and least-squares policy iteration [15] are efficient batch-mode training methods for modeling the state-action value function using function approximators. They collect samples as tuples (s_t, a_t, r_t, s_{t+1}), where s_t is the state at time t; and s_{t+1} is the next state after executing action a_t on state s_t and receiving the reward r_t, by interacting directly to the environment using either totally random actions or a ϵ-greedy policy. The dataset composed of samples are iteratively used for batch training.

This work uses a similar approach as [11]. In that work, an ESN is used to model non-Markovian environments in control tasks such as the mountain car problem and the acrobot swing-up task. It uses a dynamical system, the reservoir, to convert a non-Markovian state to a approximate Markovian state representation embedded in the high-dimensional state-space of the reservoir. Specifically, the current work uses an ESN to model partially observable environments in robot navigation problems. The experiments are performed using an e-puck robot with 8 distance sensors in a rectangular environment with an obstacle between the robot and the target region. The task of the robot is to reach the goal through the correct side of the environment. A temporary stimulus at the beginning of the episode indicates which side of the environment the robot should use. So, to successfully perform the task, the navigation system must have some sort of short-term memory.

In the proposed setup, the ESN is iteratively trained in batch mode to approximate the state-action value function given as input 8 distance sensors, the action and an additional input which simulates the temporary stimulus. Experimental results show that after an initial exploration period and a sequence of policy improvements steps, the robot is able to navigate to the goal circumventing the obstacle through the correct side (determined by the initial stimulus) in a quasi-optimal trajectory. We also show that the fading memory of the reservoir is essential to model these context-dependent navigation attractors.

This work is organized as follows. In Section 2, an overview on iterative value approximation is presented as well as the used reservoir computing architecture. Next, Section 3 introduce the experiments accomplished as well as the corresponding environments, settings, and associated results. The last section presents conclusions and future work.

2. METHODS

2.1 ITERATIVE POLICY LEARNING ON PARTIALLY OBSERVABLE ENVIRONMENTS

In fitted Q iteration [16], samples in form of tuples $(s_t, a_t, r_t, s_{t+1}), t = 1, ..., I$ 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:

$$\mathbf{u}(t) = (s_t, a_t), \tag{1}$$

$$\hat{y}(t) = r_t + \gamma \max_a \hat{Q}_{N-1}(s_{t+1}, a) \tag{2}$$

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 work, we use an Echo State Network (ESN) [7] to model the critic, that is, the Q-value [13] function, in non-Markovian environments. Given a partially observable state vector $\tilde{\mathbf{s}}$ and an action a as input, we want to approximate the expected future sum of rewards, the Q-value for the pair $(\tilde{\mathbf{s}}, a)$, using an ESN as approximation method. The randomly generated reservoir in the ESN can convert non-Markovian state-spaces into Markovian state-spaces due to its characteristic fading memory of previous inputs. It is similar to fitted Q iteration [14, 16] and least squares policy iteration [15] in that it is based on batch offline training and approximates the value function in an iterative way.

In [11], the ESN 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 ESN 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.

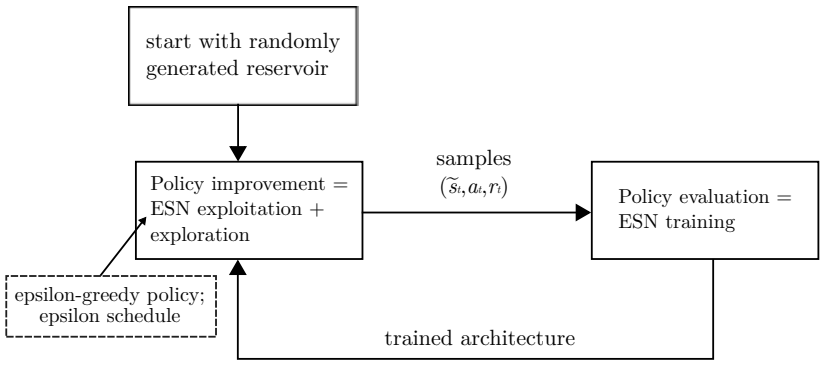


Figure 1: Approximate Policy Iteration: Policy improvement + Policy evaluation. The iterative policy learning consists of: generation of samples by interacting with the environment using a ϵ-greedy policy and the trained architecture (policy improvement); and of training the architecture (in this case, the ESN) 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 ESN architecture.

As we can approximate $\hat{Q}(\tilde{s}, a)$, the desired output $\hat{y}$, by a sum of future rewards over a finite time horizon h [11], equations (1) and (2) can be rewritten, in the case of a non-Markovian environment:

$$\mathbf{u}(t) = (\tilde{s}_t, a_t), \tag{3}$$

$$\hat{y}(t) \approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^h r_{t+h} \tag{4}$$

The training is accomplished in an iterative way and consists of a sequence of policy improvement and policy evaluation steps (see Fig. 1). 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 (3) and (4), respectively, and the ESN 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. The ESN model and its training method, linear regression, are explained in the following section. 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 [11] (given in Section 3.2).

2.2 RESERVOIR COMPUTING

An ESN is composed of a discrete hyperbolic-tangent RNN (i.e., the reservoir) and a linear readout output layer which maps the reservoir states to the desired output (Fig. 2). In this work, we use leaky integrator nodes [17] in the reservoir, whose state update equation is given by:

$$\mathbf{x}(t+1) = (1-\alpha)\mathbf{x}(t) + \alpha f(\mathbf{W}^{\text{res}}_{\text{res}}\mathbf{x}(t) + \mathbf{W}^{\text{res}}_{\text{inp}}\mathbf{u}(t) + \mathbf{W}^{\text{res}}_{\text{bias}}). \tag{5}$$

where: $\mathbf{u}(t) = [\tilde{\mathbf{s}}, a]$ denotes the input at time t, a concatenation of a non-Markovian observation $\tilde{\mathbf{s}}$ and an action a; $\mathbf{x}(t)$ represents the reservoir state; α is the leak rate [17, 18]; and $f() = \tanh()$ is the hyperbolic tangent activation function (most common type of activation function used for ESNs). The equation for the readout output $y(t)$, which models the state-action value function in this work, is as follows:

$$y(t+1) = \mathbf{W}^{\text{out}}_{\text{res}}\mathbf{x}(t+1) + \mathbf{W}^{\text{out}}_{\text{bias}}. \tag{6}$$

The weight matrices $\mathbf{W}^{\cdot}_{\cdot}$ represent the connections between the nodes of the network (where res, inp, out, $bias$ denote $reservoir$, $input$, $output$ and $bias$, respectively). All weight matrices to the reservoir (denoted as $\mathbf{W}^{\text{res}}_{\cdot}$) are initialized randomly (represented by solid arrows in Fig. 2), while all connections to the output (denoted as $\mathbf{W}^{\text{out}}_{\cdot}$) are trained (represented by dashed arrows in Fig. 2). The initial state is set to $\mathbf{x}(0) = \mathbf{0}$.

The randomly created $\mathbf{W}^{\text{res}}_{\text{res}}$ matrix is rescaled such that the system is stable and the reservoir has the echo state property (i.e., it has a fading memory [19]). This can be accomplished by rescaling the matrix so that the spectral radius $|\lambda_{max}|$ (the largest absolute eigenvalue) of the linearized system is smaller than one [19]. Standard settings of $|\lambda_{max}|$ lie in a range between 0.7 and 0.98 [20]. In this work, the reservoir weights ($\mathbf{W}^{\text{res}}_{\text{res}}$) are rescaled such that its spectral radius are $|\lambda_{max}| = 0.9$, which sets the dynamic regime of the reservoir to the edge of stability in order to achieve rich reservoir dynamics.

Next, consider the following notation: n_i is the number of inputs; n_r is the number of neurons in the reservoir; n_o is the number of outputs (in this work, $n_o = 1$); and the number of training samples is n_s. The output weights $\mathbf{W}^{\text{out}}_{\cdot}$ are trained in batch mode using standard linear regression techniques on the reservoir states. The procedure is as follows. First, the reservoir is

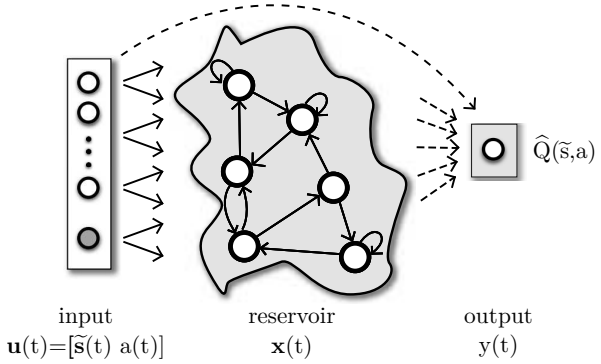
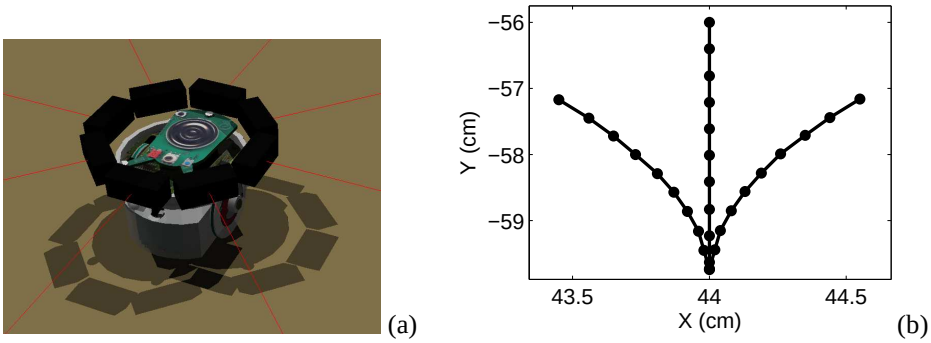


Figure 2: Reservoir Computing network as a function approximator for reinforcement learning tasks with partially observable environments. The reservoir is a dynamical system of recurrent nodes. Solid lines represent connections which are fixed. Dashed lines are the connections to be trained.



(a) (b)

Figure 3: (a) Robot model used in the experiments: e-puck extended with longer-range distance sensors. (b) Motor primitives or basic behaviors: left, forward and right.

driven by an input sequence $\mathbf{u}(1), \ldots, \mathbf{u}(n_s)$ which yields a sequence of states $\mathbf{x}(1), \ldots, \mathbf{x}(n_s)$ using (5). In this process, state noise can be added to (5) for regularization purposes, but that is not used in this work because the environment is already noisy due to wheel slippage and noisy sensors. The generated states are collected row-wise into a matrix $\mathbf{M}$ of size $n_s \times (n_r + 1)$ where the last column of $\mathbf{M}$ is composed of 1's (representing the bias). The desired teacher output, the total future expected return, is collected row-wise into a matrix $\hat{\mathbf{Y}}$. Then, the readout output's matrix $\mathbf{W}_{\text{rb}}^{\text{out}}$ (i.e., the column-wise concatenation of $\mathbf{W}_{\text{out}}^{\text{out}}$ and $\mathbf{W}_{\text{bias}}^{\text{out}}$) of size $(n_r + 1) \times n_o$ is created by solving (in the mean square sense):

$$\mathbf{M}\mathbf{W}_{\text{rb}}^{\text{out}} = \hat{\mathbf{Y}} \tag{7}$$

$$\mathbf{W}_{\text{rb}}^{\text{out}} = (\mathbf{M}^\top \mathbf{M})^{-1} \mathbf{M}^\top \hat{\mathbf{Y}} \tag{8}$$

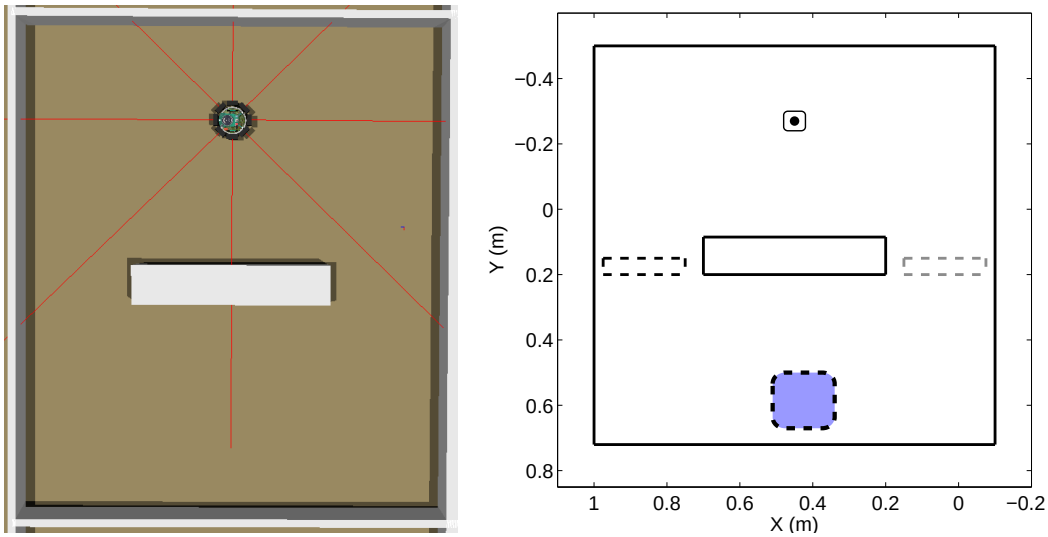
Note that the other matrices ($\mathbf{W}_{\text{res}}^{\text{res}}$, $\mathbf{W}_{\text{inp}}^{\text{res}}$, $\mathbf{W}_{\text{bias}}^{\text{res}}$) are not trained at all. Their initialization is presented in Section 3.2.

2.3 ROBOT MODEL

We use the e-puck robot from the Webots simulation environment [21] for the experiments in this work. Webots provides a physics model of the e-puck robot (the simulator detects collisions and simulates physical properties of objects, such as the mass, the velocity, the inertia, the friction, the spring and damping constants, etc.). The model is shown in Fig. 3(a). It has a 7 cm diameter. The e-puck is equipped with 8 infra-red sensors which measure ambient light and proximity of obstacles in a range of 4 cm originally. However, the extended e-puck model used in this work has longer-range sensors capable of measuring distances in the interval (0, 30] cm. The noise on sensors is drawn from a normal distribution $N(0, 3)cm$. In real-world experiments, these longer-range sensors could be implemented by adding inexpensive infra-red range sensors to the real e-puck robot via an extension module.

The robot has 2 stepper motors with maximum speed of 1000 steps per second, which are steered by the following 3 motor primitives or basic behaviors in the low-level control module: 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 ($704ms$). See Fig. 3(b) for a graphical representation of the trajectories given by each of the motor primitives. It is interesting 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.



(a) 3D environment and e-puck robot

(b) Corresponding Map in 2 dimensions

Figure 4: 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.

3. EXPERIMENTS

3.1 INTRODUCTION

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. 4(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. 4(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. 1 and explained in Sections 2.1.

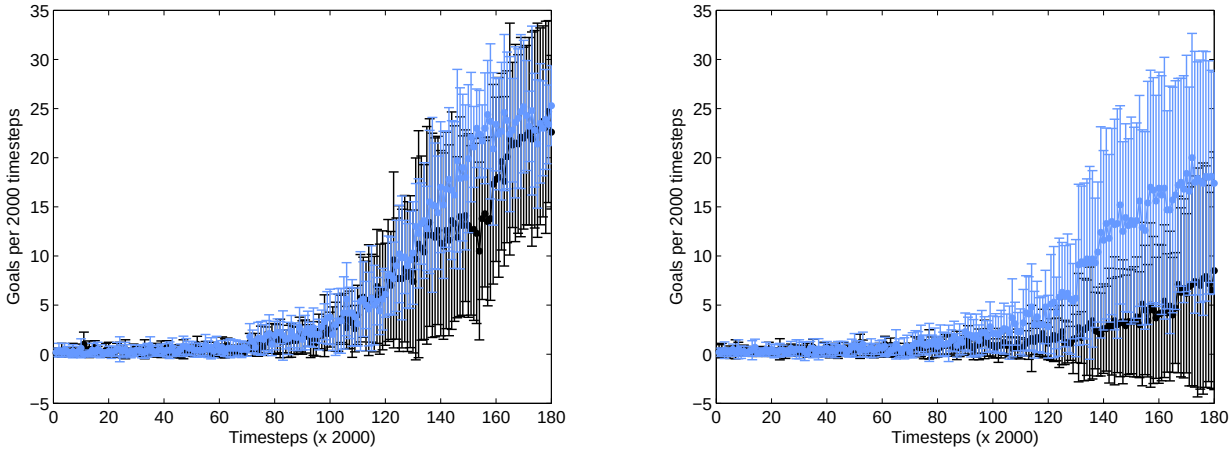
The task of the robot in this environment consists of navigating to the goal location, given by the light blue dashed box in Fig. 4(b), through the left or right part of the environment, shown by black and gray dashed rectangles in the same figure, depending on a previously received stimulus from the 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 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 (3) and (4).

3.2 SETTINGS

This section presents the configuration parameters. 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 reservoir size is 400 neurons for all experiments in this work. The leak rate is $\alpha = 0.1$. The input weight matrix $\mathbf{W}^{\text{res}}_{\text{inp}}$ is initialized to -0.14, 0.14 and 0 with probabilities 0.25, 0.25 and 0, respectively. The reservoir is sparsely connected: only 10% of the weights in $\mathbf{W}^{\text{res}}_{\text{res}}$ are non-zero, chosen from the set $\{-1, 1\}$ with equal probability [11]. The resulting matrix $\mathbf{W}^{\text{res}}_{\text{res}}$ is rescaled such that its spectral radius is $|\lambda_{max}| = 0.9$. The weights from matrix $\mathbf{W}^{\text{res}}_{\text{bias}}$ are generated from a normal distribution $N(0, 0.2)$.

The ϵ parameter for the policy, which corresponds to the probability of selecting random actions at each time step, 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 [11]. 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, $\epsilon = 0.9$. The finite time horizon in (4) is $h = 40$. The discount factor is $\gamma = 1$, which defines a shortest-path problem.



(a) ESN

(b) The same network but without recurrent connections

Figure 5: Average number of goals achieved per two thousand timesteps for 10 simulation experiments. The black lines represent the goals achieved via the *left trajectory*, while the blue lines represent the goals achieved via *right trajectory*. Error bars represent the standard deviation between runs. (a) Using the reservoir architecture presented in this chapter. (b) Using the same architecture, but without internal memory by setting $\mathbf{W}_{\text{res}}^{\text{res}} = 0$.

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.

3.3 RESULTS

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. 5(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. 5(b), the mean number of achieved goals is computed using a memoryless architecture, implemented by simply setting the reservoir weights $\mathbf{W}_{\text{r}}^{\text{r}}$ 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.

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. 6(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. 6(c) shows the principal components resulting from applying PCA on the reservoir states for the last episodes of simulation of Fig. 6(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. 6(b) 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.

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.

4. CONCLUSION

This work has shown that an Echo State Network (ESN) can be used to model the state-action value function in non-markovian navigation tasks. The training procedure is based on [11], consisting of an alternating sequence of simulation experiments for samples generation and regression learning events, under a policy iteration framework (policy improvement + policy evaluation) [15]. The ESN projects a non-Markovian input into a high-dimensional non-linear state-space with temporal dynamics. This dynamic state-space convert the non-Markovian environment into a Markovian problem, as it automatically takes the history of the input stream into account.

The non-markovian task of the robot in this work is similar to the T-maze task [22, 23] which requires a temporal association of a past stimulus and a future delayed reponse. However, the current work is more general in the sense that it requires to learn attractors from the agent-environment interaction which are capable of controlling a mobile robot to the correct goal in a partially

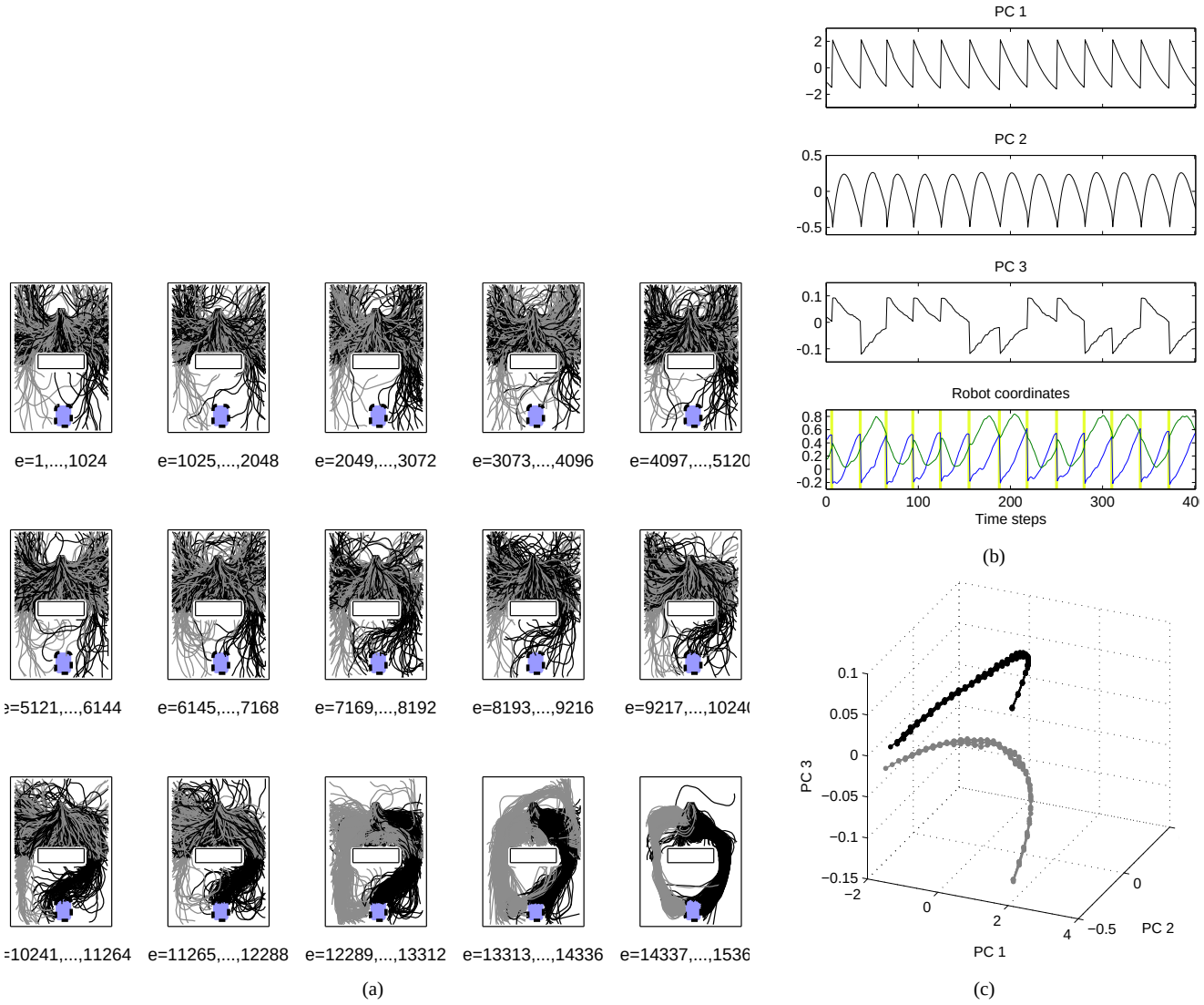


Figure 6: (a) A sequence of robot trajectories as learning evolves, using the ESN. Each plot shows robot trajectories in the environment for several episodes during the learning process. In the beginning, exploration is high and several locations are visited by the robot. As the simulation develops, 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. 6(a). The fourth plot shows the robot coordinates x, y over time in the environment. The yellow 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) 3D state space of the principal components, where gray and black represent different trajectories in the environment.

observable environment. Instead of waiting for a delayed response, the navigation task requires that the correct attractor is learned and chosen given a temporary stimulus at the beginning of the simulation.

There are several possible extensions of this work. First, experiments can be done with extended navigation tasks where the robot navigates between rooms of an environment, which would allow to evaluate how the architecture scales to more complex tasks. It would be interesting to investigate how many navigation attractors a single reservoir network can learn and how this is related to the size of the reservoir. Finally, other techniques such as Slow Feature Analysis [24] could be investigated in order to know whether they can help to infer hidden features of a robot environment from the reservoir states to increase performance and scale the experiments to bigger environments.

REFERENCES

[1] T. Bailey and H. Durrant-Whyte. "Simultaneous Localisation and Mapping (SLAM): Part II State of the Art". *Robotics and Automation Magazine*, pp. 108–117, September 2006.

[2] 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), pp. 498– 505, Vancouver, Canada, 2006.

[3] E. A. Antonelo and B. Schrauwen. "Supervised Learning of Internal Models for Autonomous Goal-oriented Robot Navigation using Reservoir Computing". In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009. (submitted).

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

[5] E. A. Antonelo and B. Schrauwen. "Towards Autonomous Self-localization of Small Mobile Robots using Reservoir Computing and Slow Feature Analysis". In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3818–3823, 2009.

[6] D. Verstraeten, B. Schrauwen, M. D'Haene and D. Stroobandt. "A Unifying Comparison of Reservoir Computing Methods". *Neural Networks*, vol. 20, pp. 391–403, 2007.

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

[8] K. Vandoorne, W. Dierckx, B. Schrauwen, D. Verstraeten, R. Baets, P. Bienstman and J. Van Campenhout. "Toward optical signal processing using Photonic Reservoir Computing". *Optics Express*, vol. 16, no. 15, pp. 11182–11192, 8 2008.

[9] D. Verstraeten, S. Xavier-de Souza, B. Schrauwen, J. Suykens, D. Stroobandt and J. Vandewalle. "Pattern classification with CNNs as reservoirs". In *Proceedings of the International Symposium on Nonlinear Theory and its Applications (NOLTA)*, 9 2008.

[10] 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.

[11] K. Bush. "An Echo State Model of non-Markovian Reinforcement Learning". Ph.D. thesis, Colorado State University, Fort Collins, CO, 2008.

[12] I. Szita, V. Gyenes and A. Loerincz. "Reinforcement Learning with Echo State Networks". In *Artificial Neural Networks - ICANN 2006*, edited by S. Kollias, A. Stafylopatis, W. Duch and E. Oja, volume 4131 of *Lecture Notes in Computer Science*, pp. 830–839. Springer Berlin / Heidelberg, 2006.

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

[14] M. Riedmiller. "Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning methods". In *In 16th European Conference on Machine Learning*, pp. 317–328. Springer, 2005.

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

[16] D. Ernst, P. Geurts and L. Wehenkel. "Tree-Based Batch Mode Reinforcement Learning". *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, 2005.

[17] H. Jaeger, M. Lukosevicius and D. Popovici. "Optimization and Applications of Echo State Networks with Leaky Integrator Neurons". *Neural Networks*, vol. 20, pp. 335–352, 2007.

[18] 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 International Conference on Artificial Neural Networks (ICANN)*, 2007.

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

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

[21] O. Michel. "Webots: Professional Mobile Robot Simulation". *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[22] E. A. Antonelo, B. Schrauwen and D. Stroobandt. "Mobile Robot Control in the Road Sign Problem using Reservoir Computing Networks". In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2008.

[23] C. Ulbricht. "Handling Time-Warped Sequences with Neural Networks". In *From Animals to Animats 4: Proc. Fourth Int. Conf. on Simulation of Adaptive Behaviour*, edited by e. a. Maes P., pp. 180–192. MIT Press, 1996.

[24] L. Wiskott and T. J. Sejnowski. "Slow Feature Analysis: Unsupervised Learning of Invariances". *Neural Computation*, vol. 14, no. 4, pp. 715–770, 2002.