The 19th European Conference on Artificial Intelligence (ECAI 2010)

Proceedings of the

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)

Monday, August 16 2010
Lisboa, Portugal

Nils T Siebel, Josef Pauli and Yohannes Kassahun

http://www.erlars.org/

Proceedings of the 3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Editors: Nils T Siebel, Josef Pauli and Yohannes Kassahun

ISSN 2190-5576 (Print)
ISSN 2190-5746 (Internet)

Published and Printed by
Nils T Siebel
Wilmsstr. 5
10961 Berlin
Germany
1st Edition, August 2010

Table of Contents

A Message from the Chairs

We would like to welcome you to the 3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems, ERLARS 2010, held in conjunction with the ECAI 2010 conference in Lisboa, Portugal on August 16 2010.

The ERLARS workshop is concerned with research on efficient algorithms for evolutionary and reinforcement learning methods to make them more suitable for autonomous robot systems. The long term goal is to develop methods that enable robot systems to learn completely, directly and continuously through interaction with the environment. In order to achieve this, methods are examined that can make the search for suitable robot control strategies more feasible for situations in which only few measurements about the environment can be obtained.

The articles contained in these proceedings are steps along this way. We hope that they can serve as a useful set of ideas and methods to achieve the long term research goal. In order to give researchers a chance to discuss their work at an early stage this proceedings volume also includes short papers / research statements.

We would like to thank the program committee members who provided very good and helpful reviews. We are also especially indebted to the authors of the articles sent to this workshop for providing the material to make us think and discuss.

It has been a great pleasure organising this event and we are happy to be supported by such a strong team of researchers. We sincerely hope that you enjoy the workshop and we look forward, with your help, to continue building a strong community around this event in the future.

Nils T Siebel, Josef Pauli and Yohannes Kassahun, Chairs, ERLARS 2010 Workshop.

Organisation of the ERLARS 2010 Workshop

Workshop Chairs

Nils T Siebel	Josef Pauli	Yohannes Kassahun
Building Automation Lab	Intelligent Systems Group	Research Group Robotics
Department of Engineering I	Department of Computer Science	DFKI Lab Bremen
HTW University of Applied Sciences	University of Duisburg-Essen	University of Bremen
Berlin, Germany	Duisburg, Germany	Bremen, Germany

Programme Committee

Andrew Barto, Autonomous Learning Laboratory, University of Massachusetts Amherst, USA.

Peter Dürr, Laboratory of Intelligent Systems, EPFL Lausanne, Switzerland.

Christian Igel, Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany.

Takanori Koga, Computational Brain Science Laboratory, Yamaguchi University, Japan.

Tim Kovacs, Department of Computer Science, University of Bristol, UK.

Jun Ota, Graduate School of Engineering, University of Tokyo, Japan.

Jan Peters, Max Planck Institute for Biological Cybernetics, Tübingen, Germany.

Daniel Polani, Department of Computer Science, University of Hertfordshire, Hatfield, UK.

Marcello Restelli, Artificial Intelligence and Robotics Laboratory, Politecnico di Milano, Italy.

Stefan Schiffer, Department of Computer Science, RWTH Aachen University, Germany.

Juergen Schmidhuber, Swiss AI Lab IDSIA, Lugano, Switzerland.

Sergiu-Dan Stan, Technical University of Cluj-Napoca, Romania.

Jeremy Wyatt, School of Computer Science, University of Birmingham.

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Lisboa, Portugal, August 16 2010

Learning Adaptive Navigation Strategies for Resource-constrained Systems

Armin Hornung[1] and **Maren Bennewitz**[1] and **Cyrill Stachniss**[1] and **Hauke Strasdat**[2] and **Stefan Oßwald**[1] and **Wolfram Burgard**[1]

Abstract. The majority of navigation algorithms for mobile robots assume that the robots possess enough computational or memory resources to carry out the necessary calculations. Especially small and lightweight devices, however, are resource-constrained and have only restricted capabilities. In this paper, we present a reinforcement learning approach for mobile robots that considers the imposed constraints on their sensing capabilities and computational resources, so that they can reliably and efficiently fulfill their navigation tasks. Our technique learns a policy that optimally trades off the speed of the robot and the uncertainty in the observations imposed by its movements. It furthermore enables the robot to learn an efficient landmark selection strategy to compactly model the environment. We describe extensive simulated and real-world experiments carried out with both wheeled and humanoid robots which demonstrate that our learned navigation policies significantly outperform strategies using advanced and manually optimized heuristics.

1 INTRODUCTION

Completing navigation tasks reliably and efficiently is one of the most essential objectives for an autonomous robot. As a precondition for finding the way to a target location, the robot needs to know its pose in the environment. Especially in the case of small robots with a limited payload, such as humanoids or unmanned aerial vehicles, compact and lightweight cameras are often the only available sensor for navigation. However, the movements of a mobile robot typically introduce motion blur in the acquired images, with the amount of degradation depending on camera quality, on the lighting conditions, and on the movement velocity. Figure 1 depicts two images of patches of a wooden floor recorded with a downward-looking camera on a wheeled robot and a humanoid robot walking through the same corridor. As can be seen, the movements of the robots introduce substantial motion blur to the image, which in practice will lead to a considerable reduction of the accuracy of the position estimation process. While there are methods to reduce the influence of motion blur [22] or limit image acquisition to stable phases of a gait [13], the degradation introduced by motion blur usually cannot be completely eliminated by filtering techniques and cheap cameras typically do not allow for an exact synchronization to the controllers executing the motor commands or the walking gait.

Additionally, small humanoids or unmanned aerial vehicles are often resource-constrained and possess only limited computational power. For truly autonomous navigation in initially unknown environments, however, the robot has to solve the so-called simultaneous

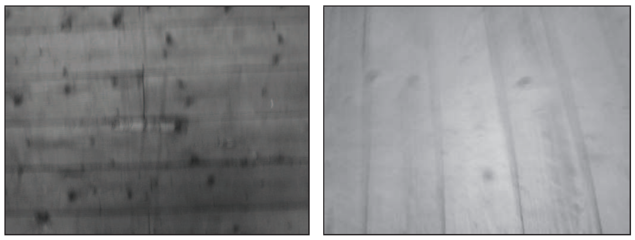


Figure 1. An indoor floor patch observed by a wheeled robot moving at 0.4 m/s (left), and by a walking humanoid robot (right). Significant motion blur is introduced in the captured images, degrading their quality for feature detection.

localization and mapping (SLAM) problem. This is computationally demanding and the memory requirements increase with the number of landmarks that need to be maintained by the robot. In practice, there are many scenarios in which the number of visible landmarks during a navigation task is significantly larger than the number of landmarks that can be processed efficiently on an embedded device. This leads to the question which landmark should be stored and maintained by the robot to optimally solve the navigation task.

In general, the goal of the robot is to accomplish its task as fast as possible. However, as faster movements may introduce a higher uncertainty in the pose estimate due to the decreased reliability of the sensor data, they increase the risk of not being able to accomplish the mission. In principle, the robot therefore has to determine the movement speed that provides the optimal trade-off between the time needed to reach a designated target location and the risk of a positioning failure. Such questions often arise on systems with limited computational resources. The systems are usually not able to incorporate all the information into the state estimation processes which introduces a corresponding information selection problem. In this paper, we present a general approach towards learning optimal policies for systems with limited computational or perceptional capabilities, which at the same time are efficient and lead to reliable navigation behaviors. We use reinforcement learning (RL) to learn which navigation actions to execute so as to reach the destination reliably and efficiently. At each time step, the robot decides whether it should decrease the velocity or even stop to increase the quality of its perceptions or to continue moving towards the goal. In previous publications on vision-based navigation with wheeled and humanoid robots [9, 10, 21], we discussed the key concepts of our approach. Besides the localization problem, we investigate in this work how reinforcement learning can be used to decide which landmark to in-

[1] Dept. of Computer Science, University of Freiburg, Germany
[2] Department of Computing, Imperial College London, UK

tegrate during navigation without a known map [28]. We present experiments carried out in simulation and with real wheeled and humanoid robots and demonstrate that the learned policies significantly outperform manually optimized strategies and also techniques using advanced heuristics.

This paper is structured as follows. We first give an overview over related work in Sec. 2, followed by the background about state estimation and reinforcement learning in Sec. 3. Sec. 4 details our learning approach. Finally, in Sec. 5 we present the experimental results.

2 RELATED WORK

In the last few years, various frameworks have been presented which employ active methods in the context of localization and navigation. Kollar and Roy [15] use reinforcement learning to optimize the robot's trajectory during exploration. Similar to our approach, the authors learn optimal parameters of the navigation controller. While we consider the problem of reaching the destination reliably and as fast as possible, Kollar and Roy learn the translational and rotational behavior which minimizes the uncertainty in SLAM (simultaneous localization and mapping). Huynh and Roy [12] generate control laws by combining global planning and local feedback control to obtain trajectories which minimize the pose uncertainty during navigation. Cassandra *et al.* [5] introduced *dual-mode controllers* as heuristics for POMDPs. A threshold on the entropy as a measure of the uncertainty determines whether a greedy action or an action reducing the uncertainty is selected.

A different method of minimizing the uncertainty about the state of the robot is to plan a path for the robot which takes the information gain into account. A popular approach in this context is the so-called *coastal navigation* introduced by Roy *et al.* [23]. Recently, He *et al.* [8] have applied this technique to a quadrotor helicopter for indoor navigation with a short-range laser range finder.

Michels *et al.* [18] proposed to learn a control policy for high speed obstacle avoidance of a remotely controlled car. Based on depth estimation with a monocular vision system, steering directions are learned. The authors focus on obstacle avoidance whereas we consider the effect of fast movements on the observation quality and adapt the speed accordingly. Kwok and Fox [16] apply reinforcement learning to increase the performance of soccer-playing robots by active sensing. In their approach, the robot learns where to point its camera to localize relevant objects.

Bennewitz *et al.* [3] developed a localization method based on visual features and presented experiments with a humanoid robot. The authors mentioned the impact of motion blur on feature extraction, but did not address the problem specifically. Instead, their robot interrupted its movement at fixed intervals to make observations. To overcome the problem of motion blur in the context of humanoid robots, Ido *et al.* [13] explicitly consider the shaking movements of the head while walking and acquire images only during stable phases of the gait.

Pretto *et al.* [22] proposed an additional image processing step prior to feature extraction, in particular for humanoid robots. The authors estimate the direction of the motion blur for image patches and present a novel feature detection and tracking scheme. While their approach increases the matching performance, motion blur cannot be completely removed by filtering. However, such a pre-processing technique could be easily combined with our learning approach to further improve the navigation performance of the robot.

Miura *et al.* [19] presented a method for adaptive speed control in partially unknown environments. In this approach, the velocity is chosen to be as fast as possible while still being safe in the sense that potential collisions with obstacles are avoided. The authors use heuristics which depend on the distance of the robot to unexplored areas and empirically determined safety margins around obstacles.

In this paper, we do not only investigate the ability to localize a vehicle but also to build maps under constraint settings. The standard method for SLAM relies on the extended Kalman filter (EKF) [6] or its variants such as the unscented Kalman filter (UKF) [14]. Using these approaches, the computational requirement and memory demand increase at least quadratically with the number of landmarks since the full correlation between the position of all landmarks is taken into account. There are many approximative filtering techniques for SLAM [20, 30]. These methods do not incorporate the full correlation between the landmarks, so that the computational constraints are less restrictive. However, their memory demand increases at least linearly with the number of landmarks used.

Recently, Sala *et al.* [25] presented a graph-theoretic formulation for the selection problem of visual features to perform navigation in known environments. The optimal set of features is defined as the minimal set with which navigation is possible. Zhang *et al.* [32] proposed an entropy-based landmark selection method for SLAM. This method specifies a measure of which visible landmark is best in terms of entropy reduction. However, it only provides a vague guideline for how many features should be selected at a given point in time. Furthermore, Lerner *et al.* [17] presented another quality measure for landmark selection in known environments which is based on the comparison of pose uncertainties. Dissanayake *et al.* [6] suggested a map management which ensures a uniform distribution of landmarks over the traversed area. Apart from landmark selection, other active methods were presented such as maximizing the SLAM estimate by intelligent path planning [4].

3 BACKGROUND

3.1 The Unscented Kalman Filter

The *unscented Kalman filter* (UKF) is a recursive Bayes filter to estimate the state $\mathbf{x}_t$ of a dynamic system [14]. This state is represented as a multivariate Gaussian distribution $N(\mu, \Sigma)$. The estimate is updated using nonlinear controls and observations $\mathbf{u}_t$ and $\mathbf{z}_t$. The key idea of the UKF is to apply a deterministic sampling technique that is known as the unscented transform to select a small set of so-called sigma points around the mean. Then, the sigma points are transformed through the nonlinear state transition and measurement probability functions, and the Gaussian distributions are recovered from them thereafter. The UKF can better deal with non-linearities and thus leads to more robust estimates compared to other techniques such as the extended Kalman filter.

3.2 Vision-based Localization

In this work, we use the UKF to perform state estimation. In case of localization, it estimates the 3D pose of the robot in a given 2D map of the environment. Besides Monte Carlo localization, Kalman filter-based localization is one of the standard techniques applied in mobile robotics.

A control $\mathbf{u}_t$ for the UKF is obtained from the robot's motion. On wheeled robots, an odometry motion model can be used, utilizing the data from the robot's wheel encoders [31]. Humanoid robots can use the executed motion command as a rough guess, or estimate their movement by integrating the leg joint angles while walking [11].

As observations $\mathbf{z}_t$, we extract *speeded-up robust features* (SURF) [2] from the camera images. Extracted descriptors of these features are then matched to landmarks in a map. This was constructed beforehand for each environment and contains the global 2D positions and SURF descriptors of the landmarks on the floor. Whenever the robot matches a perceived feature to a landmark in the map, it integrates the relative 2D position of the landmark as observation $\mathbf{z}_t = (r_t, \varphi_t)$ in the UKF in order to estimate its pose $\mathbf{x}_t = (x_t, y_t, \theta_t)$.

3.3 Simultaneous Localization and Mapping

We also use the UKF for the setting when the environment is not known to the robot and the positions of landmarks need to be estimated as well. This problem is widely known as the landmark-based simultaneous localization and mapping (SLAM) problem where one seeks to simultaneously determine the map of the environment and the pose of the robot. We apply the UKF as a probabilistic method to estimate the joint probability distribution over the robot's pose and the landmark locations:

$$p(\mathbf{x}_t, \mathbf{l}_1, \ldots, \mathbf{l}_M \mid \mathbf{u}_1, \ldots, \mathbf{u}_t, \mathbf{z}_1, \ldots, \mathbf{z}_t) \tag{1}$$

Here, $\mathbf{x}_t$ is the pose of the robot at time t and the position of the landmarks $\mathbf{l}_1, \ldots, \mathbf{l}_M$ given all previous motions $\mathbf{u}_1, \ldots, \mathbf{u}_t$ and observations $\mathbf{z}_1, \ldots, \mathbf{z}_t$. Various approaches to estimate this posterior have been presented in the literature.

In this paper, we address the SLAM problem using the UKF by representing the joint state $(\mathbf{x}_t, \mathbf{l}_1, \ldots, \mathbf{l}_M)$ with $\langle \mu, \Sigma \rangle$. This is a standard approach which has been shown to operate successfully in the past. The mean of the jth landmark location (μ_{2j+2}, μ_{2j+3}) is denoted by $\left(l_x^{[j]}, l_y^{[j]}\right)$. Furthermore, we interpret the state transition function as the robot's motion model and assume that range and bearing observations (r, φ) are given so that we can define a corresponding observation model.

3.4 Reinforcement Learning

In reinforcement learning, an agent seeks to maximize its reward by interacting with the environment [29]. Formally, this is defined as a *Markov decision process* (MDP) using the state space $\mathcal{S}$, the actions $\mathcal{A}$, and the rewards $\mathcal{R}$. By executing an action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$, the agent experiences a state transition $s_t \rightarrow s_{t+1}$ and obtains a reward $r_{t+1} \in \mathcal{R}$. The overall goal of the agent is to maximize its return R_t given by

$$R_t = \sum_{i=t+1}^{T} r_i \ , \tag{2}$$

where T is the time when the final state is reached. One finite sequence of states $s_0, \ldots, s_T$ is called an *episode*.

The decision of which action to take in a certain state is governed by the policy

$$\pi(s, a) = p(a|s) \ \ \forall s \in \mathcal{S} \ , \tag{3}$$

which denotes the probability of taking action a in state s. The *action-value function*, also called *Q-function*, for a policy π is defined as

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} \ , \tag{4}$$

which denotes the expected return of taking action a in state s and following policy π afterward. The optimal policy maximizes the expected return, which corresponds to the maximum Q-value for each state-action pair.

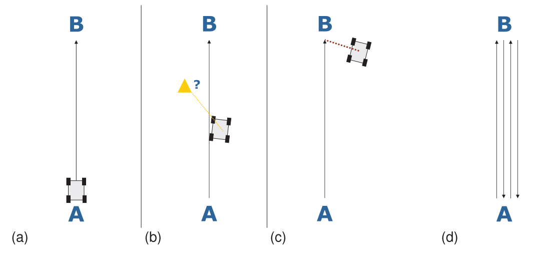


Figure 2. Illustration of the single-goal navigation task (a-c) and the round-trip task (d).

4 LEARNING NAVIGATION POLICIES

In our work, we consider three typical navigation tasks and analyze how to solve them in the reinforcement learning setting. In the first task, the robot has to reach a target location as fast as possible while staying localized using its camera and a given map. To achieve this, the robot has to adapt its travel speed to obtain good feature observations on the one hand, while it has to drive as fast as possible to reach its goal quickly on the other hand. Note that this single goal navigation task can be easily extended to a multi-waypoint path following task. Such a path of waypoints may be given to the robot by a higher level task planner or a path planner such as A^*.

The second and third task address the problem of navigating in an environment without a known map towards a given (relative) location and to perform round-trip navigation tasks, respectively. Here, the challenge is to select a good subset of landmarks to solving the SLAM problem while at the same time taking into account the computational constraints of the system.

Because the belief about the robot's state is represented by a probability distribution in the UKF, the system is ideally modeled by a partially observable MDP (POMDP) [27], which requires an explicit modeling of the probability distribution of the state. This makes POMDPs computationally hard to solve and intractable for most real-world tasks. We use the so-called *augmented MDP* [24] as approximation of the POMDP. Hereby, the belief of the state is represented by its most-likely estimate and the task is modeled as an MDP. The uncertainty of the underlying belief distribution is taken into account by including the corresponding entropy in the state representation.

4.1 Navigation Tasks

4.1.1 Navigation Task with a Given Map

Let us consider the following most basic navigation task (Fig. 2(a)). The robot is located at a starting position A and is supposed to reach a goal position B. In this first setting, the robot is supposed to have a map of the environment, that means it knows where the individual landmarks it can observe are located in the environment. However, the robot's motion is affected by drift and the overall motion influences the visual perception of the robot because the observed scene is affected by motion blur. The faster the robot moves, the more its visual perception is degraded. This has a direct impact on feature extraction and matching and, thus, on the localization performance. By moving slowly or stopping from time to time, the negative impact of motion blur can be avoided, but the robot needs more time to finish the navigation task.

Rewards Since we require the robot to reach the goal as quickly as possible, we encode this directly in the reward function. The immediate reward at time t is given as

$$r_t = \begin{cases} C & \text{if } t = T \\ -\Delta_t & \text{otherwise,} \end{cases} \tag{5}$$

where C is some constant, T is the final time step, and Δ_t is the time interval between the update steps. The final state is reached when the robot's true pose is sufficiently close to the destination. This has the effect that the robot is driven to reach the destination as fast as possible in order to maximize its reward.

Note that we do not model an explicit punishment for delocalization or running into a wall. We assume that the robot has some sensors for obstacle avoidance on board, such as bumpers, infrared, or sonar. When the robot is in danger of running into an obstacle, it is immediately stopped by the obstacle avoidance. The time it takes to stop, re-localize, and accelerate is the implicit punishment for getting off the track, which is typically a few seconds.

Actions The set of available actions in this task is coupled to the mobile robot at hand. On our wheeled robot, we have a basic navigation controller available which steers the robot to the next goal point based on the current most-likely pose estimate $\mathbf{x}_t = (x_t, y_t, \theta_t)$ and the desired target velocity v_{target}. Depending on the angle ϕ to the next goal point, the translational and rotational velocities v and ω are set in the following way. When $|\varphi| \geq \frac{\pi}{2}$, v is set to zero and the robot orients itself towards the goal. Otherwise, v is set to the desired target velocity v_{target} and ω is set depending on ϕ.

As parameter of the navigation controller which influences the quality of the observed images, we learn the overall velocity limit v_{target} as combination of the translational and rotational velocity. That means that the resulting actions for reinforcement learning can be kept as simple as setting v_{target} (in m/s) to the following values:

$$\mathcal{A} = \{0.1, 0.2, 0.3, 0.4, 1.0\} \; . \tag{6}$$

Regarding the humanoid robot, a discrete set of actions can be directly used to learn the controlling policy to reach the goal fast and reliably. This eliminates the need for a navigation controller that steers the robot to the goal, because the full controller policy is learned. Note that there is still a gait controller running on the humanoid, which translates the walking commands into commands for the joint angles. On our humanoid, we employ the following actions:

- Walk forward: The robot walks 10 cm in forward direction (2 steps).
- Turn left / turn right: The robot turns $23°$ on the spot in the given direction (2 steps).
- Stand still: The robot interrupts its movement and waits for 0.7 seconds to acquire a good quality image for its localization. This is the time required for the robot's body to stabilize after it has stopped.

We chose these actions since they proved to yield the most reliable and predictable behavior.

4.1.2 Single-goal Navigation Task Without Known Landmark Locations

In the second scenario, the landmark locations are not known to the robot and it also has to reach a given location (specified in relative coordinates to the start location). Thus, the robot has to solve a similar task as before but without a map and thus has to estimate the map online (Fig. 2(b-c)) . Here, the problem arises that estimating the map as well leads to a significantly increased overhead in memory and computational load. Thus, the key task of the robot is to select and integrate only landmarks that are useful for the navigation task. We assume that N landmarks are distributed randomly over the environment. When the robot perceives a new landmark, it has to decide whether it should integrate this landmark in the UKF or not. The UKF has a landmark capacity of M landmarks with $M \ll N$.

Rewards The goal is to choose the landmarks in such a way that the distance of the final position of the robot $(x_T, y_T)^\top_{\text{true}}$ and the target position B is minimized. In this scenario, we ignore the impact of the robot's velocity on its perception and the potential problem of missing landmark detections due to motion blur for now. Hence, we define the reward as

$$r_t = \begin{cases} -\left|B - (x_T, y_T)^\top_{\text{true}}\right| & \text{if } t = T \\ 0 & \text{else,} \end{cases} \tag{7}$$

which is the negative Euclidean distance of the robot's true position to the goal B if the training episode reaches the terminal state s_T; intermediate rewards are set to zero. In this task, the terminal state is reached when the robot's estimated position is at the goal B.

Actions In this task, we utilize the existing navigation controller described above with a constant velocity. Thus, the robot only needs to decides whether to integrate a new landmark or not, which is a binary decision:

$$\mathcal{A} = \{a_{\text{reject}}, a_{\text{accept}}\} \tag{8}$$

4.1.3 Round-trip Task Without Known Landmark Locations

In the round-trip task, the robot is supposed to reach several subgoals (see Fig. 2 (d)). It starts at A and is supposed to drive to B, back to A and then drive to B and A again. A new subgoal is selected as soon as the position estimate of the robot $(x_t, y_t)^T$ is close to the current subgoal – independent of the robot's true position $(x_t, y_t)^T_{\text{true}}$. In this task, the error in the pose estimate should be minimized over the whole trajectory. For convenience, we specify the return directly as the negative average error over the remaining trajectory

$$R_t = -\frac{1}{|T-t|} \sum_{t'=t}^{T} \left| \begin{pmatrix} x_{t'} \\ y_{t'} \end{pmatrix}_{\text{true}} - \begin{pmatrix} x_{t'} \\ y_{t'} \end{pmatrix} \right| , \tag{9}$$

whereas t specifies the current time and T is the time when the robot reaches its final destination. The actions are identical to the single-goal SLAM task. To simplify things for the second task, landmark selection is only allowed while the robot moves from A to B the first time. The round-trip task is more complex than the previous one. However, it is worth considering since it focuses on the loop-closing problem of SLAM where a robot re-visits previously seen areas in order to correct incremental pose errors. Therefore, this task has a higher practical relevance than the single-goal task.

4.2 State Space $\mathcal{S}$

The complete state of the robot consists of the global pose estimate $\mathbf{x}_t$, the current velocity, and a characterization of the environment including landmarks and waypoints to reach. However, this complete state representation is impractical to consider for reinforcement learning. Learning in this complete description would take too long and generalization would be hard to achieve.

Thus, we define a set of features based on the complete state which characterizes the state sufficiently detailed and as general as needed for learning a specific navigation task. Based on the current, most-likely pose estimate $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$ and the environment, we define the following features:

- The Euclidean distance to the next goal point $(g_x, g_y)^\top$

$$d = \sqrt{(g_x - x_t)^2 + (g_y - y_t)^2}. \tag{10}$$

- The angle relative to the next goal point

$$\phi = \operatorname{atan2}(g_y - y_t, g_x - x_t) - \theta_t. \tag{11}$$

  In combination with d, this completely characterizes the relative position of the next goal point which has to be reached. In multiple-waypoint scenarios, the next waypoint is regarded as goal point.
- The uncertainty of the localization, represented in terms of the differential entropy of the pose:

$$h = \frac{1}{2} \ln \left((2\pi e)^3 \; \left| \det \left(\Sigma^{3\times 3} \right) \right| \right). \tag{12}$$

  This measures how well the robot is localized: A higher entropy corresponds to a higher pose uncertainty.

In addition, the following features are relevant in the context of landmark integration in SLAM:

- The angle φ_l to the potential new landmark $l^{[\text{new}]}$.
- The number of landmarks already integrated in the UKF

$$m = \left| \{ j \in M : \Sigma_{2j+2} < \infty \wedge \Sigma_{2j+3} < \infty \} \right|, \tag{13}$$

  where Σ_{2j+2} and Σ_{2j+3} are the variances of the jth landmark in the x and y direction.
- The distance of the potential new landmark to the closest landmark already integrated

$$d_l = \min_{\substack{j \in L \text{ with} \\ \Sigma_{2j+2} < \infty \wedge \Sigma_{2j+3} < \infty}} \left| \begin{pmatrix} l_x^{[j]} \\ l_y^{[j]} \end{pmatrix} - \begin{pmatrix} l_x^{[\text{new}]} \\ l_y^{[\text{new}]} \end{pmatrix} \right|. \tag{14}$$

For the localization task with motion blur (see Sec. 4.1.1), we found the features d, ϕ, and h to be most relevant and sufficient for completing the task. Other combinations of them, also including the current velocity and the landmark density in the state representation, did not lead to a significant improvement of the robot's performance.

In the single-goal and round trip SLAM tasks, we use a combination of all of the above features, and additionally evaluate the effectiveness of including the entropy in the state space.

Since the state space of the features is usually continuous, we need to estimate the Q-function with some function approximator. Either k-nearest neighbor (k-NN) regression [26] or radial basis function (RBF) networks [7] yielded good results in our experiments. In contrast to a strictly discrete representation as feature table, these methods suffer less from the effects of discretization.

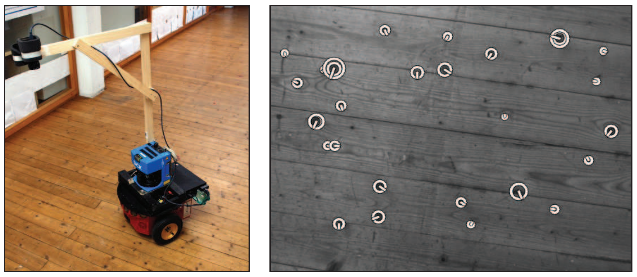


Figure 3. Pioneer 2-DX8 robot in the experimental indoor environment (left) and an observed floor patch with SURF as visual landmarks (right).

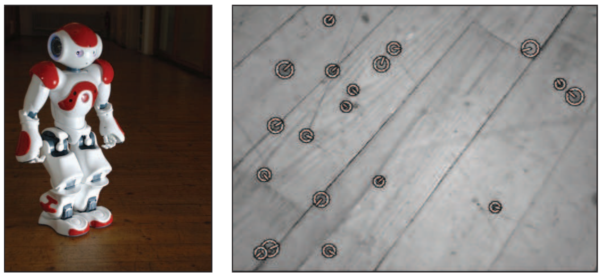


Figure 4. The *Nao* humanoid robot [1] in the experimental indoor environment (left) and an observed floor patch with SURF as visual landmarks (right).

5 EXPERIMENTS

5.1 Navigation Policy for a Known Map

We evaluated our approach for known environments on a wheeled *Pioneer* robot (Fig. 3) as well as on our *Nao* humanoid robot (Fig. 4). The wheeled robot was equipped with a top-mounted camera observing the floor in front of it. Additionally, it carries a laser range finder for obstacle avoidance and to provide a ground truth pose estimate for evaluation.

The humanoid robot is equipped with two small cameras of webcam quality. One of them points to the ground in front of the robot, which we use for localization. In addition, Nao has two ultrasound sensors which can be used for obstacle avoidance. Since the humanoid has no knowledge about its true pose, we use a special marker to allow the robot to identify when it has reached the goal location. This artificial landmark can be reliably detected even while the robot is walking.

5.1.1 Learning the Navigation Policy in Simulation

The policy was learned in simulations. This allowed us to evaluate different parameter settings for the learning algorithms and to run a large number of learning and testing episodes without putting too much strain on the real robots. Each simulated robot and its environment are modeled as close to reality as possible. This includes the motion noise of the robots with a systematic drift to the left or right. We use a map of artificial landmarks whose positions are randomly distributed. To avoid an adaption of a robot's behavior to a specific environment, landmark positions and the direction of the systematic motion error were randomized in each new learning and evaluation episode.

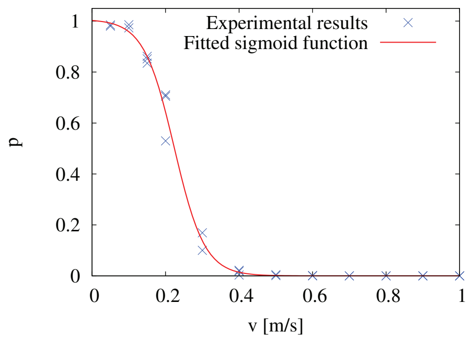


Figure 5. Experimentally determined observation model $p(z|v)$ for a given velocity v of a wheeled robot in our indoor environment. The measured data (blue crosses) are approximated by a sigmoid function (red line).

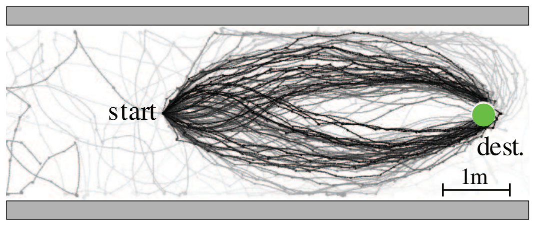


Figure 6. Evolution of the trajectory followed by the humanoid throughout the 1500 learning episodes (each 10th episode is drawn). At the beginning, random exploratory actions are chosen and the robot does not reach the goal within a maximum of 700 seconds (light gray trajectories). Towards the end, the robot navigates successfully and efficiently towards the destination (black trajectories). Note that there is a high noise in the executed motion commands.

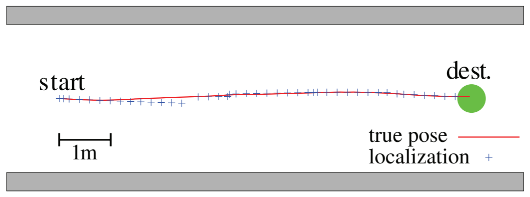


Figure 7. A typical example of the executed trajectory of a learned policy. The corresponding state space is displayed in Fig. 8.

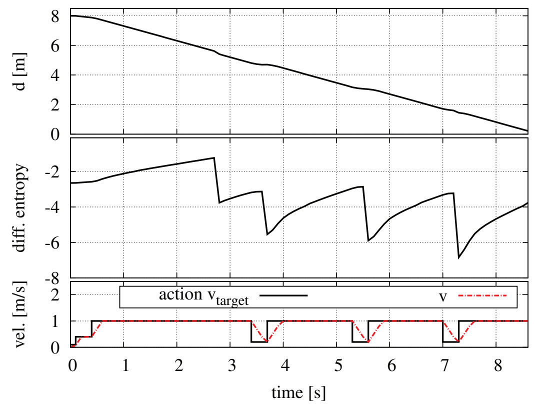


Figure 8. A typical example of the learned policy for a wheeled robot with the dimensions *distance* and *entropy* of the state space. The corresponding trajectory is displayed in Fig. 7. The robot maximizes its velocity until its uncertainty gets too high, indicated by a high value of the differential entropy. To re-localize, it then slows down. As soon as the uncertainty decreases as an effect of localization, it accelerates again. As the robot approaches the goal location, it slows down more frequently.

In order to obtain a policy which takes motion blur into account, we modeled motion blur in the simulation environment as an effect on the probability of an observation z given the current velocity v, i.e., we determined the probability that a feature which is in the robot's field of view is detected given v. This dependency $p(z|v)$ was experimentally estimated using real data and is approximated by a sigmoid function (Fig. 5). On the humanoid, the amount of motion blur depends on the executed motion command and the current phase of the walking cycle. As we are not able to accurately synchronize the image acquisition with the walking cycle, we use the average observation probability for each walking motion instead.

Note that we model motion blur as effect on the observation probability only in the simulation environment, it is not part of the learning state space or the robot's state estimate. Instead, the robot learns about this effect while interacting with the environment.

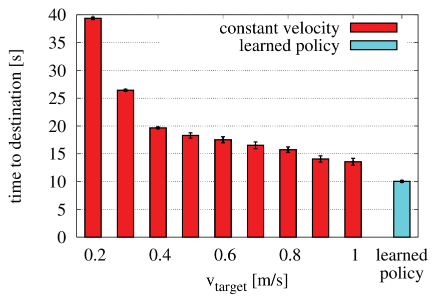
Figure 6 shows the evolution of the humanoid's behavior throughout the learning process. As can be seen, in the beginning the robot chose random exploratory actions. It did not reach the goal so that the episodes were aborted after a maximum of 700 seconds (the resulting trajectories are colored light gray). After a certain number of learning trials, however, the robot successfully navigated towards the destination (dark gray / black trajectories). The trajectories were getting more and more efficient towards the end of the learning process. Note that the robot had a systematic error in the executed motion command in each of the episodes.

5.1.2 Evaluation of the Learned Policy

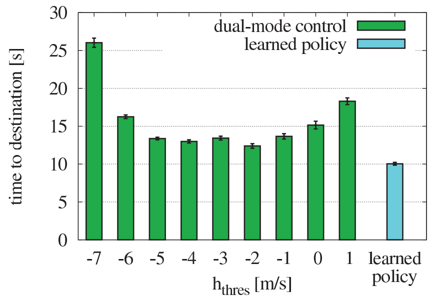
A typical trajectory and the corresponding state space over time of the learned policy for a wheeled robot are displayed in Fig. 7 and 8. The robot optimizes its time to reach the destination by driving at maximum speed as long as it is confidently localized. When there is risk of getting lost, indicated by a high entropy, it slows down in order to observe landmarks. Note that for different values of the distance d, different levels of the entropy are learned to be important. As the robot gets closer to the goal, it frequently slows down so that the target is reliably reached. Overall, the robot stays close to the direct connection between start and destination.

Comparison to Constant Velocity A standard approach for a wheeled robot is to set a constant target velocity v_{target}. Figure 9(a) displays an evaluation of following a constant velocity from $v_{\text{target}} =$ 0.2 m/s to 1 m/s, compared to our learned policy. Up to 0.4 m/s, an increased velocity directly improves the time to destination. For higher velocities, the robot is no longer able to perform observations, regularly gets lost on its path, and has to stop in order to avoid collisions and to re-localize. Despite this, there is still a small improvement in the average time to destination. This means the robot accepts the risk of nearly colliding and getting lost in favor of a faster speed.

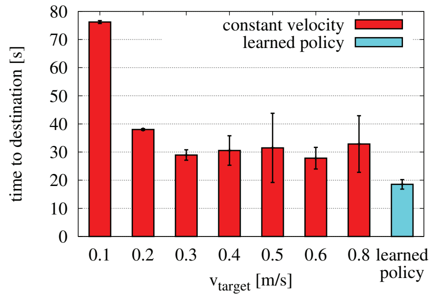
But even when choosing the best policy of constant velocity, our learned approach is significantly better. While the average time to destination at 1 m/s is 13.56 s $\pm$ 0.61 s (95% confidence interval),



(a) Constant velocity in the simulated scenario (100 runs).



(b) Dual-mode control policies at various thresholds for the entropy in the simulated scenario (100 runs).



(c) Constant velocity in the real indoor scenario (10 runs).

Figure 9. Comparison of constant velocity policies and the dual-mode controller to our learned policy in known environments. Each policy is displayed with mean and 95% confidence interval. The learned policy is significantly better than each other policy.

the robot is able to finish the task with our learned policy in 10.04 s $\pm$ 0.18 s, which corresponds to a reduction of 26%.

Comparison to Dual-Mode Controllers A more advanced approach is to employ a *dual-mode controller* as introduced by Cassandra *et al.* [5]. Similar to our learned policy, the entropy is used to decide on which action to take. When the entropy is above a threshold h_{thres}, an action to reduce the uncertainty is selected, otherwise a greedy action is chosen. These actions are $v_{\text{target}} = 0.1$ and $v_{\text{target}} = 1.0$ in our scenario, respectively. Figure 9(b) displays the resulting times for various values of h_{thres} compared to the learned policy on the wheeled robot.

Using the dual-mode controller, we achieve best results for $h_{\text{thres}} = -2$, resulting in a time to reach the destination of 12.39 s $\pm$ 0.31 s. The learned policy still yields a significant reduction of 17%.

Additionally, we evaluate the performance of a policy learned using our approach for the humanoid robot in comparison to a dual-mode control policy. This dual-mode policy controls the robot to walk forward while the estimated orientation towards the goal is smaller than some threshold $\hat{\varphi}$. Whenever the estimated angular distance to the goal is larger than $\hat{\varphi}$, the robot stops its forward motion and turns towards to goal. We chose the threshold of $\hat{\varphi} = 35^\circ$ since smaller values lead to an oscillating behavior of the robot near the destination, whereas larger values lead to frequent collisions with the walls bounding the corridor. Whenever the uncertainty about its pose exceeds a threshold, the robot stops in order to obtain a good quality image. This threshold was empirically determined to minimize the time to reach the destination while still achieving a success rate of 100%. Thus, we optimized the parameters of this dual-mode controller so as to perform best in our test environment.

It took the humanoid robot 117.18 s $\pm$ 8.41 s to reach the destination with the hand-optimized controller, and only 106.66 s $\pm$ 10.05 s using our learned policy. A t-test with 95% confidence reveals that the learned policy performs significantly better.

5.1.3 *Verification on Real Robotic Systems*

We now transfer the results from simulations into the real world by applying the policy learned in simulation on real robots.

Wheeled Robot We first employ the Pioneer robot (Fig. 3) in an indoor environment. Each policy is evaluated in 10 test runs, each

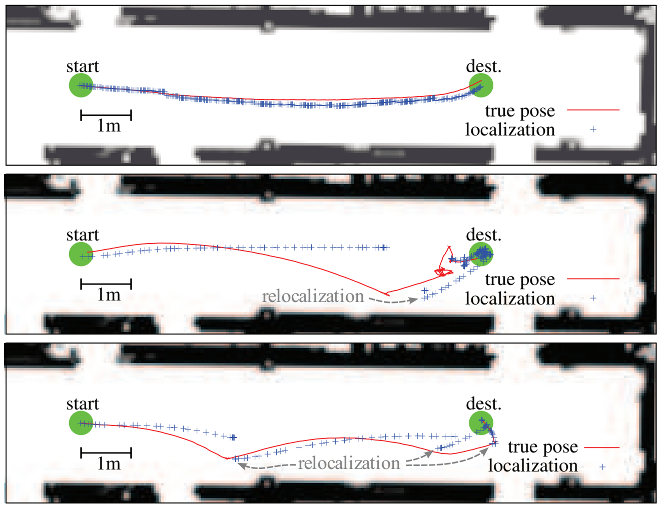


Figure 10. Comparison of real robot trajectories at constant velocity (top: 0.2 m/s, middle: 0.8 m/s) and variable velocity, i.e., following the learned policy (bottom).

consisting of navigating from the start location to the destination. The resulting navigation times are shown in Fig. 9(c).

Similar to the results from simulations, the learned policy outperforms any policy of constant velocity by more than 25% and is significantly better. When looking at the trajectories generated by the policies qualitatively, the results are also similar to the simulated ones (Fig. 10). At a slow constant velocity, the robot stays close to the optimal path of the straight-line connection between start and destination. When driving faster at 0.8 m/s, the robot is not able to observe landmarks and quickly gets lost with the result of a near-collision with the wall. Note that there is a systematic drift to the right in the robot's motion. Contrary to that, the robot does not need to be stopped by the obstacle avoidance while following the learned policy. When it in danger of getting lost, it immediately slows down to re-localize. As a result, the robot reaches its destination reliably and quickly.

Humanoid Robot Finally, we performed experiments with our real humanoid (Fig. 4) navigating in our hallway environment. We

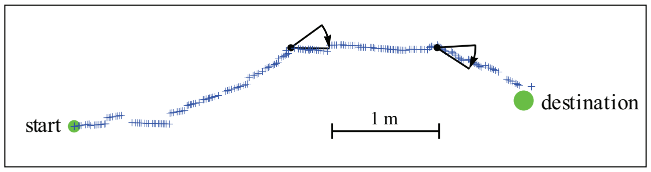


Figure 11. Estimated trajectory of the humanoid while executing the learned navigation policy in our hallway. The robot walks forward with an error resulting from a drift to the left. Whenever it seems appropriate according to its belief, the robot executes a turning action to re-align with the goal. The robot stops as soon as it recognizes the goal landmark.

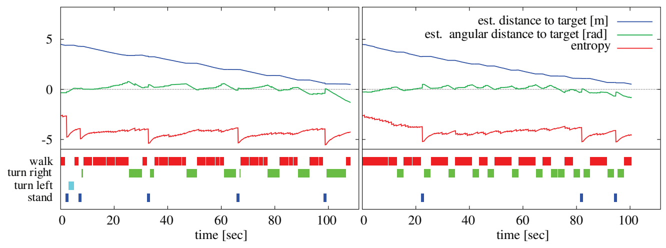


Figure 12. The state features over time during typical runs with a hand-optimized controller (left) and the learned navigation policy (right) in simulation. The hand-optimized controller stops the robot in regular intervals to decrease the uncertainty, whereas the learned policy adapts the observation frequency according to the current state.

conducted test runs both with the hand-optimized controller and with the policy learned in simulation. The robot needs 93.16 s $\pm$ 10.14 s using the hand-optimized controller compared to 86.53 s $\pm$ 8.60 s using the learned policy, so the learned policy outperforms the hand-optimized controller by 7.1%. Again, a t-test with 95% confidence shows that the learned policy performs significantly better.

Figure 11 depicts a typical trajectory of the Nao robot while executing the learned navigation strategy (the drawn poses were estimated by the localization system). As can be seen, the robot rotates from time to time to compensate for its motion drift and to re-align with the goal. This learned policy is not adapted to the specific drift direction.

Note that in these experiments, we used slow walking patterns as the Nao's stability was highly reduced when walking faster in the current implementation. Accordingly, the acquired images are only moderately blurred. The robot can still match an average of 3.25 features per frame while moving, and actions to reduce the uncertainty are rarely executed. Thus, the efficiency gain of the learned controller compared to the hand-optimized controller results mainly from choosing the navigation actions more foresightedly, which leads to shorter paths.

In future implementations, we will optimize the humanoid's gait, so faster walking patterns will be used. While this will enable the humanoid to potentially reach its goal faster, it also increases the amount of motion blur, thus seriously reducing the average number of successfully matched features while moving. To evaluate the impact of motion blur, we learned policies for a different set of estimated observation probabilities in the simulator, i.e., we decreased the probability of a successful feature match by 80% during walking and turning.

Again, we compared the learned policy to a hand-optimized dual-mode controller that stops the humanoid whenever the entropy exceeds a fixed threshold. For each of the different observation probabilities, we selected the hand-optimized controller leading to the smallest average time to destination while still achieving a success rate of 100 %. The results show that the learned policy is significantly faster (9% gain) than the hand-optimized policy.

Figure 12 shows the state space of two typical runs with the hand-optimized controller (left image) and the learned policy (right image). The hand-optimized controller stops the robot in regular intervals to obtain good observations. In contrast to that, the learned policy accepts higher uncertainties as long as the distance to the destination is high, whereas it increases the robot's stand frequency when approaching the destination.

5.2 Landmark Selection Policy for Navigation in Unknown Environments

We evaluate the performance of our learned landmark selection policies in the single-goal and round trip SLAM scenarios on our wheeled *Pioneer* platform (Fig. 3), first in simulations and then on the real robot.

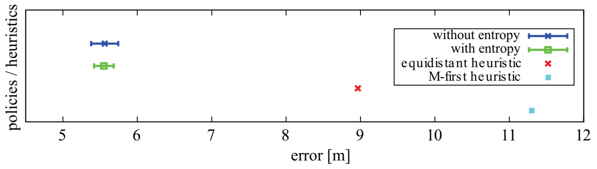


Figure 13. Average performance of the learned policies and heuristics w.r.t. 1,000 test episodes in the single-goal SLAM task. For the learned policies, the mean over ten training runs as well as the corresponding 95%-confidence interval is shown.

5.2.1 *Single-goal Task in Simulation*

For the single-goal task in simulation, we choose an environment where N landmarks are randomly distributed in a 30 m by 60 m area. The distance between the start position A and the goal B is set to 44 m. We train our policy for 1,000 episodes. In each episode, landmarks are randomly re-distributed. We compare the trained policies with two heuristics. The first one is the *M-first heuristic* which simply integrates the M first landmarks that are observed. An apparently better policy is the *equidistant heuristic*. With this heuristic, the robot only integrates a new landmark after it has driven a certain distance so that the landmarks are approximately uniformly distributed over the whole trajectory (similar to [6]).

At first, we consider an UKF with a landmark capacity of $M = 10$ and an environment with $N = 50$ landmarks. For each learning approach, ten training runs are performed. Each trained policy and heuristic is evaluated in 1,000 different environments (see Fig. 13). The one-sample t-test at 95% confidence shows that all three learning approaches are significantly better than the equidistant heuristic.

A notable fact is that in this setting, we were not able to show that there is any benefit from including the feature of the entropy h of the robot's pose in the state space. Even at 75% confidence, the t-test did not reveal a difference between the learning approach using the entropy compared to the setting where it is ignored. One reason why the current entropy of the robot's pose is not a good indicator of whether to integrate a landmark or not in the SLAM task is that landmarks are integrated with an uncertainty over the robot's pose. That means that the robot is not able to reduce its uncertainty immediately after integrating a landmark. The relative position of the landmark, for example, is a better indicator on how the robot will perform in reaching the goal.

In order to evaluate how good the trained policies generalize, we trained and tested a policy in environments with $N = 50$ as well as $N = 100$ landmarks. In addition, we use UKFs with a capacity M of five, ten, and 15 landmarks. Fig. 14 (a) illustrates the high degree of generalization of our learning approach. For instance, if we perform a training in a setting with $N = 50$ and $M = 5$, we see that the trained policy leads to significantly better results than the equidistant heuristic in all six test scenarios. This indicates that our approach generalized over different landmark densities which is similar to environments of different scale and sensor range.

5.2.2 *Single-goal Task Performed in a Real World Experiment*

Furthermore, we evaluated our landmark selection learning approach in the real experimental environment. Similarly to Sec. 5.1, we use a pioneer robot equipped with a camera and laser range finder in a hallway environment. Learning the policy in this real-world environment would be impractical because this would not only require us to perform hundreds of training episodes but also to install different landmark distributions for each training episode. Thus, we trained the policy in simulation and tested it in the real-world setting. We also compared the trained policy to the equidistant heuristic. Both the trained policy as well as the equidistant heuristic were tested ten times. The trained policy results in an error of 0.50 ± 0.08 m whereas the equidistant heuristic leads to an error of 0.66 ± 0.07 m. Hence, the trained policy is significantly better than the equidistant heuristic (w.r.t. a t-test at 95% confidence).

5.2.3 *Round-trip Task*

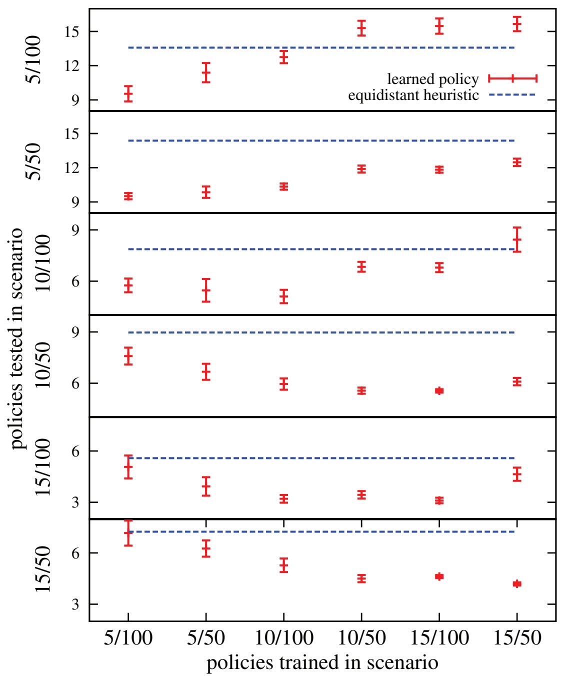
The performance of our learning procedure for the round-trip task is evaluated in a simulated environment with a wheeled robot, similar to the single-goal task. The error evaluation, however, differs since the average localization error over the whole trajectory was considered here to provide a better performance when approaching also the intermediate goal. Again, we compare our learning with the equidistant heuristic. Fig. 14 (b) shows that the learned policy is significantly better than the heuristic. Furthermore, it is shown that we were able to generalize over the UKF capacity M as well as the number of landmarks N.

6 CONCLUSION

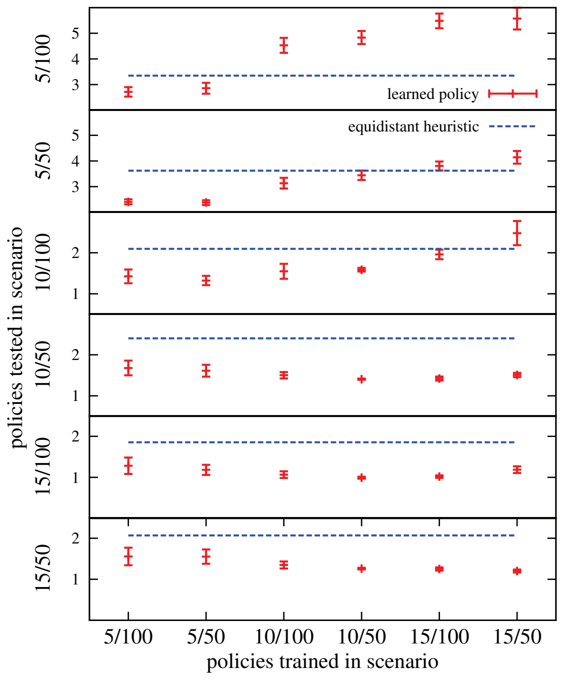
In this paper, we presented a novel approach to learning efficient navigation policies for mobile robots that are constrained in their sensing capabilities as well as in their computational resources. We considered navigations tasks in known as well as unknown environments. By considering these navigation problems as reinforcement learning tasks, the robot can learn policies for choosing appropriate actions.

In case of navigating in known environments, the robot is able to select the optimal velocity so that it reaches its target location as fast as possible and with minimum error.

For navigation in unknown environments, the map has to be estimated as well to navigate efficiently. This task, however, requires significant computational resources. We presented an approach to learn



(a) Single-goal task



(b) Round-trip task

Figure 14. High degree of generalization in the single-goal task (a) and the round trip task (b) in unknown environments. The mean error over ten training runs and the corresponding standard derivation is shown. All policies below the dashed horizontal line are significantly better than the equidistant heuristic ($\alpha = 0.05$).

an efficient landmark selection policy. The ability of a mobile robot to decide which landmark to incorporate into its belief given the navigation task at hand allows for navigation under computational constraints. The presented method is able to determine which landmark is valuable for the robot to efficiently solve its current navigation task.

In a series of real-world and simulated experiments with wheeled robots and a humanoid, we demonstrated that our learned navigation policies significantly outperform strategies using advanced and manually optimized heuristics.

ACKNOWLEDGEMENTS

This work has been supported by the German Research Foundation (DFG) under contract number SFB/TR-8 and within the Research Training Group 1103.

REFERENCES

[1] Aldebaran Robotics. The Nao humanoid robot. http://www.aldebaran-robotics.com/en/. Retrieved June 2010.

[2] H. Bay, T. Tuytelaars, and L. V. Gool, 'SURF: Speeded-up robust features', *Proc. of the ninth European Conf. on Computer Vision*, (2006).

[3] M. Bennewitz, C. Stachniss, W. Burgard, and S. Behnke, 'Metric localization with scale-invariant visual features using a single perspective camera', in *European Robotics Symposium 2006*, ed., H. Christiensen, volume 22 of *STAR Springer tracts in advanced robotics*, (2006).

[4] M. Bryson and S. Sukkarieh, 'Active airborne localisation and exploration in unknown environments using inertial SLAM', in *Proc. of the IEEE Aerospace Conference*, (2006).

[5] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien, 'Acting under uncertainty: Discrete bayesian models for mobile-robot navigation', in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (1996).

[6] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, 'A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem', in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'00)*, pp. 1009–1014, (2000).

[7] K. Doya, 'Reinforcement learning in continuous time and space', *Neural Computation*, **12**(1), 219–245, (2000).

[8] R. He, S. Prentice, and N. Roy, 'Planning in information space for a quadrotor helicopter in a GPS-denied environments', in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, (2008).

[9] A. Hornung, M. Bennewitz, and H. Strasdat, 'Efficient vision-based navigation – Learning about the influence of motion blur', *Journal of Autonomous Robots*, **29**, 137–149, (August 2010).

[10] A. Hornung, H. Strasdat, M. Bennewitz, and W. Burgard, 'Learning efficient policies for vision-based navigation', in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (2009).

[11] A. Hornung, K. M. Wurm, and M. Bennewitz, 'Humanoid robot localization in complex indoor environments', in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (2010). Accepted for publication.

[12] V. A. Huynh and N. Roy, 'icLQG: Combining local and global optimization for control in information space', in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, (2009).

[13] J. Ido, Y. Shimizu, Y. Matsumoto, and T. Ogasawara, 'Indoor Navigation for a Humanoid Robot Using a View Sequence', *The International Journal of Robotics Research*, **28**(2), 315–325, (2009).

[14] S. J. Julier and J. K. Uhlmann, 'A new extension of the Kalman filter to nonlinear systems', in *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, pp. 182–193, (1997).

[15] T. Kollar and N. Roy, 'Using reinforcement learning to improve exploration trajectories for error minimization', in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, (2006).

[16] C. Kwok and D. Fox, 'Reinforcement learning for sensing strategies', in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (2004).

[17] R. Lerner, E. Rivlin, and I. Shimshoni, 'Landmark selection for task-oriented navigation', *IEEE Transaction on Robotics*, **23**(3), (2007).

[18] J. Michels, A. Saxena, and A. Y. Ng, 'High speed obstacle avoidance using monocular vision and reinforcement learning', in *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pp. 593–600, New York, NY, USA, (2005). ACM.

[19] J. Miura, Y. Negishi, and Y. Shirai, 'Adaptive robot speed control by considering map and motion uncertainty', *Journal of Robotics & Autonomous Systems*, **54**(2), 110–117, (2006).

[20] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, 'FastSLAM: A factored solution to the simulaneous localization and mapping problem', in *Proc. of the National Conf. on Artificial Intelligence (AAAI'02)*, pp. 593 – 598, (2002).

[21] S. Oßwald, A. Hornung, and M. Bennewitz, 'Learning reliable and efficient navigation with a humanoid', in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, (2010).

[22] A. Pretto, E. Menegatti, M. Bennewitz, W. Burgard, and E. Pagello, 'A visual odometry framework robust to motion blur', in *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, (2009).

[23] N. Roy, W. Burgard, D. Fox, and S. Thrun, 'Coastal navigation–mobile robot navigation with uncertainty in dynamic environments', in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, (1999).

[24] N. Roy and S. Thrun, 'Coastal navigation with mobile robots', in *Advances in Neural Processing Systems 12 (NIPS)*, volume 12, (1999).

[25] P. Sala, R. Sim, A. Shokoufandeh, and S. Dickinson, 'Landmark selection for vision-based navigation', *IEEE Transaction on Robotics*, **22**(2), (2006).

[26] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, MIT Press, Cambridge, MA, USA, 2006.

[27] E. J. Sondik, *The optimal control of partially observable Markov decision processes*, Ph.D. dissertation, Stanford University, Stanford, USA, 1971.

[28] H. Strasdat, C. Stachniss, and W. Burgard, 'Which landmark is useful? Learning selection policies for navigation in unknown environments', in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, (2009).

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

[30] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, 'Simultaneous localization and mapping with sparse extended information filters', *Int. Journal of Robotics Research*, **23**, (2004).

[31] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, The MIT Press, September 2005.

[32] S. Zhang, L. Xie, and M. Adams, 'Entropy based feature selection scheme for real time simultaneous localization and map building', in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'05)*, (2005).

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Lisboa, Portugal, August 16 2010

Locomotion gait optimization for a quadruped robot

Miguel Oliveira [1], **Cristina Santos** [2], **Manuel Ferreira** [3], **Lino Costa** [4] and **Ana Rocha** [5]

Abstract. This article describes the development of a gait optimization system that allows a fast but stable robot quadruped crawl gait.

We focus in the development of a quadruped robot walking gait locomotion that combine bio-inspired Central Patterns Generators (CPGs) and Genetic Algorithms (GA). The CPGs are modelled as autonomous differential equations, that generate the necessary limb movement to perform the walking gait, and the Genetic Algorithm perform the search of the CPGs parameters.

This approach allows to explicitly specify parameters such as amplitude, offset and frequency of movement and to smoothly modulate the generated trajectories according to changes in these parameters. It is therefore easy to combine the CPG with an optimization method. A genetic algorithm determines the best set of parameters that generates the limbs movements. We intend to obtain a walking gait locomotion that minimizes the vibration and maximizes the wide stability margin and the forward velocity.

The experimental results, performed on a simulated Aibo robot, demonstrated that our approach allows low vibration with a high velocity and wide stability margin for a quadruped walking gait locomotion.

1 Introduction

Robot locomotion is a challenging task that involves the control of a great number of degrees of freedom (DOF's). Several previous works, [11, 22, 15], proposed biologic approaches to modulate the gait locomotion of quadruped robots, combining biometric sensory information with motion oscillators such as CPGs.

The problem of finding the best possible locomotion is a problem currently addressed in the literature [2, 6, 14]. Usually optimization systems are applied to improve the performance of the Aibo quadruped robot locomotion. The competition in Robocup is one of the motivation engines, for these works. In the following, we briefly describe some relevant works in this domain.

In [2] it is presented a Genetic Algorithm robust to the noise in the parameters evolution and that also avoids premature local optima. The evaluation is made on a robot soccer field, and the robot communicates by wireless with an external computer where the learning algorithm is executed. The goal of the fitness is to maximize the robot velocity. As a result of this learning algorithm the robot moves with a velocity of 0.290 (m/s).

A comparison between several learning locomotion algorithms, including Genetic Algorithm and Policy Gradient Algorithm, is presented in [14]. This is also an online learning performance that uses three Aibo robots for decreasing the time spending on each test. The optimization goal is to determine the best 12 parameters of an elliptical locus scheme of locomotion, such that the robot takes less time to walk a certain distance. Each learning has a previous hand-tuned set of parameters and the best results were achieved the by the hill climbing and Policy Gradient Algorithm. The average speed achieved by the Policy Gradient Algorithm was 0.291 (m/s).

In [6] it is presented an evolutionary algorithm based on a genetic algorithm. The genetic operators are chosen by an adaptation mechanism. The locomotion is implemented in real time and it is evaluated by analyzing the forward-backward motion, the side-walk, the rotation motion and the vibration. For measuring the vibration they use accelerometers of the robot. For each sensor and during a test the standard deviation (std) of accelerometer measurements is calculated. The evaluation tests are performed in a robot soccer field, and the evaluation calculus is made in an external computer using an external camera for motoring the translation and rotational movements.

In [10] it is presented an evolutionary algorithm (EA) to optimize a vector of parameters for locomotion of an ERS110 robot. The EA uses a steady-state algorithm that applies the mutation and/or the recombination of operators to create new individuals from sets of parents. To avoid local minima an individual can be a parent during a predefined number of times. The solution obtained by the EA moves the robot with a velocity of 0.167 m/s.

In [13] it is presented an optimization system for the locomotion of Aibo 210 based on the Powell's method. It optimizes 12 parameters of a locus locomotion scheme. Optimization is made online. In each iteration the robot moves between two landmarks, and the goal is to maximize the forward velocity. They achieved an average speed of 0.2269 (m/s) for the rectangular locus and an average speed of 0.25 (m/s) for the trapezoid locus.

In this work, we propose an approach to optimize a walk gait locomotion, using Central Pattern Generators (CPGs) and a genetic algorithm.

CPGs are neural networks located in the spine of vertebrates, that generate coordinated rhythmic movements, namely locomotion [8]. In this work, a locomotion controller, based on CPGs, generates trajectories for hip robot joints [15]. These CPGs are modelled as coupled oscillators and solved using numeric integration. They have been previously applied in drumming [4] and postural control [1].

The proposed CPG is based on Hopf oscillators, and allows to explicitly and smoothly modulate the generated trajectories according to changes in the CPG parameters such as amplitude, offset and frequency. In order to achieve the desired walk gait movement, it is necessary to appropriately tune these parameters. In this work, these

[1] Industrial Electronics Department, School of Engineering,University of Minho, Portugal, email: mcampos@dei.uminho.pt
[2] Industrial Electronics Department, School of Engineering,University of Minho, Portugal, email: cristina@dei.uminho.pt
[3] Industrial Electronics Department, School of Engineering,University of Minho, Portugal, email: mjf@dei.uminho.pt
[4] Production Systems Department, School of Engineering, University of Minho, Portugal, email: lac@dps.uminho.pt
[5] Production Systems Department, School of Engineering, University of Minho, Portugal, email: arocha@dps.uminho.pt

parameters are optimized using a Genetic Algorithm. Optimization is done online in a simulated ers-7 AIBO robot using Webots [17].

This optimization is a non-linear problem where continuity and convexity conditions are not guaranteed. Thus, searching for a global optimum is a difficult task that requires approaches based on stochastic algorithms like evolutionary algorithms, in particular, genetic algorithms. These are search algorithms that mimic the process of natural selection [5]. Thus, unlike conventional algorithms, in general, only the information regarding the objective to optimize is required. Moreover, they are based on a population that evolves over time, possibly in the direction of the optimum.

This article is structured as follows. In Section 2, we explain how we generate locomotion . Section 3 presents the optimization system and it is discussed the objective function. Simulated results are described in Section 4. The paper ends with a discussion and conclusions in Section 5.

2 LOCOMOTION GENERATION

In this section we describe the system used to generate locomotion. Firstly, a brief description of gaits focusing in the generated gait is done. Next, a description of the modelled CPGs is done, including the network of Hopf oscillators.

2.1 Gait Description

During locomotion, quadruped walking animals have to usually move their legs in a manner that provides the suitable forward force at a minimal energy expenditure while maintaining their equilibrium. This coordinated cyclic manner of lifting and placing the legs on the ground, called a gait, is important for equilibrium stability and the step cycle sequence is typical for vertebrates: left forelimb (LF), right hindlimb (RH), right forelimb (RF), and left hindlimb (LH).

Quadrupedal gaits are classified according to the duration of their stance phases [16], i.e. their duty factor values, and their relative phases. In general, the duty factor β reduces as the speed increases.

In this work we will address a crawl gait. This is a symmetric gait, meaning that the two legs of the same girdle are 0.5 out of phase. This gait is singular (two or more legs are simultaneously lifted or placed during a stride) and regular (all the legs have the same duty factor).

In general, the number of step cycles per second increases as the speed of locomotion increases [7]. This corresponds to a reduction in the step cycle duration almost exclusively due to a shortening of the stance phase (limb in contact with the ground), whereas the swing phase (no ground contact) is kept nearly constant.

Herein, we assume that at all walking speeds the onset of swing in a foreleg occurs just after the onset of stance in the ipsilateral hind leg [7]. In order to achieve this, we use the wave gait rule: the gait phase (φ_{LH}) follows the value of the duty factor (β). The use of this rule improves the stability of the locomotion [9, 16, 12]. Stability is measured by calculating the stability margin [9] which decreases approximately linearly with the velocity increase (see results).

2.2 Rhythmic Movement Generation

The rhythmic movements of each hip joint of a limb, i, are generated by a Hopf oscillator, given by

$$\dot{x}_i = \alpha\left(\mu - r_i^2\right)(x_i - O_i) - \omega z_i, \tag{1}$$

$$\dot{z}_i = \alpha\left(\mu - r_i^2\right) z_i + \omega (x_i - O_i), \tag{2}$$

where $r_i = \sqrt{(x_i - O_i)^2 + z_i^2}$, amplitude of the oscillations are given by $A = \sqrt{\mu}$, ω specifies the oscillations frequency (in rad s^{-1}) and relaxation to the limit cycle is given by $\frac{1}{2\alpha\mu}$.

This oscillator contains an Hopf bifurcation from a stable fixed point at $(x_i, z_i) = (O_i, 0)$ (when $\mu < 0$) to a structurally stable, harmonic limit cycle, for $\mu > 0$.

The following expression for ω allows an independent control of speed of the ascending and descending phases of the rhythmic signal [18], meaning an independent control of the stance ω_{st} and the swing durations ω_{sw},

$$\omega = \frac{\omega_{st}}{1 + e^{-qz_i}} + \frac{\omega_{sw}}{1 + e^{-qz_i}}. \tag{3}$$

The stance phase frequency, ω_{st}, is determined based on the constant swing frequency, ω_{sw}, and on the desired duty factor, β as follows:

$$\omega_{st} = \frac{1-\beta}{\beta}\omega_{sw} \tag{4}$$

Each CPG takes a set of parameters for the modulation of the generated trajectories for the specified joint. These are:

- μ, switches on/off the rhythmic solution, and for $\mu > 0$ modulates the amplitude of oscillations;
- β, changes the walking velocity since it controls the stance duration of the generated movement.

All these parameters will be tuned by the optimization system described in section 3, controlling the parameters for a locomotion that maximizes a fitness. The parameters α, ω_{sw} and a are set a priori. Parameter ω_{sw} specifies the swing phase duration, which is kept constant. Its value depends on the desired speed of movements and on the capabilities of the robotic platform.

2.3 Interlimb Coordination

We have four CPGs, one for each Hip joint. These four CPGs are coupled in order to achieve the limb coordination required in a walking gait pattern. The applied coupling scheme is depicted in fig 1 and is given by

$$\begin{bmatrix}\dot{x}_i\\ \dot{z}_i\end{bmatrix} = \begin{bmatrix}\alpha\left(\mu - r_i^2\right) & -\omega\\ \omega & \alpha\left(\mu - r_i^2\right)\end{bmatrix}\begin{bmatrix}x_i - O_i\\ z_i\end{bmatrix} + \sum_{j\neq i}\mathbf{R}(\theta_i^j)\begin{bmatrix}x_j - O_j\\ z_j\end{bmatrix}. \tag{5}$$

The linear terms are rotated onto each other by the rotation matrix $\mathbf{R}(\theta_i^j)$, where θ_i^j is the required relative phase among the i and j hip oscillators to perform the gait (we exploit the fact that $\mathbf{R}(\theta) = \mathbf{R}^{-1}(-\theta)$).

The final result is a network of oscillators with controlled phase relationships, able to generate more complex, synchronized behavior such as locomotion. Due to the properties of this type of coupling among oscillators, the generated trajectories are stable and smooth and thus potentially useful for trajectory generation in a robot.

The generated x_i solution of this nonlinear oscillator is used as the control trajectory for a Hip joint of the robot limbs. These trajectories encode the values of the joint's angles (°) and are sent online for the lower level PID controllers of each limb joint. The knee joints are controlled as simple as possible. When the limb performs the swing phase, the knee flexes to a fixed angle. When performing the stance phase, the knee extends to other angle.

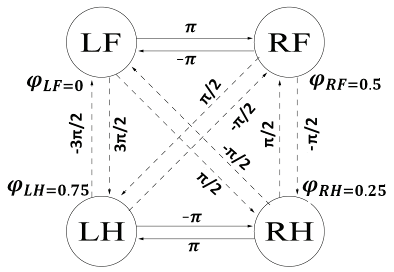


Figure 1. Each leg lags one quarter of a cycle, in the sequence: Left Fore, Right Hind, Right Fore, Left Hind

3 OPTIMIZATION SYSTEM

In this section, we explain how the limbs trajectories are optimized, in order to obtain the best walking pattern locomotion. We intend to maximize the velocity and wide stability margin, and to minimize the vibration of the robot. A scheme of the optimization system is depicted in fig 2.

CPGs generate trajectories for the robot limbs, modulated according to a set of parameters. A chromosome is constituted by the required set of these parameters such that the robot performs the desired locomotion gait.

Initially, a random initial population of chromosomes is generated. After the evaluation of all chromosomes of the population, a genetic algorithm generates a new population to be tested. The stopping criterion of the optimization system is the performed number of iterations.

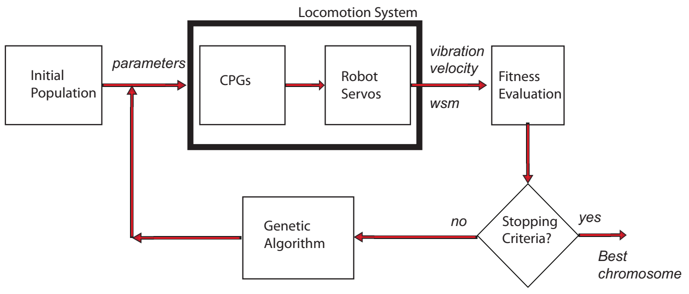


Figure 2. Optimization Locomotion System

3.1 Optimization of Parameters

As previously described, trajectories are generated and modulated by the proposed network of CPGs, by explicitly changing the CPGs parameters. These are the Amplitude (μ), the Offset(O), and the stance knee value, for each limb. Further, there is the swing frequency (ω_{sw}) for the overall network. Meaning a total of 13 parameters.

Fore and hind limb trajectories (LF, RF) (LH, RH) have the same amplitude, offset and frequency but a different relative phase. they have a relative phase of π among them.

Taking in regard these considerations, we can minimize the number of parameters it is required to optimize. The set of parameters is given by: amplitude of the front limbs (μ_{FL}), amplitude of the hind limbs (μ_{HL}), front limbs knee angle (K_{FL}),hind limbs knee angle (K_{HL}), front limbs offset (O_{FL}), hind limbs offset (O_{HL}) and the frequency of Swing (ω_{sw}). This yields a total of 7 parameters to tune.The limits of each parameters are defined in tab 1.

3.2 Limits of the parameters

The range of each parameter is defined in tab 1. These boundaries directly depend on the physics limits of the Aibo Ers-7 robot. The values of μ_{FL} and μ_{HL} are limited by the maximum range that the AIBO Hip joints may have. Noteh that amplitude is given by $\sqrt{\mu}$. Offset values O_{FL} and O_{HL} for the hips are limited by the same ranges and the calculated amplitude values, μ_{FL} and μ_{HL}, respectively .

We calculate the maximum and minimum values for each knee stance angle, such that leg collision does not occur during locomotion. This is given by

$$K_{FLmax1} = -(O_{FL} + \sqrt{\mu_{FL}}/2) + 50 \tag{6}$$
$$K_{FLmax2} = -(O_{FL} - \sqrt{\mu_{FL}}/2) + 50 \tag{7}$$
$$K_{FLmin1} = -(O_{FL} + \sqrt{\mu_{FL}}/2) + 20 \tag{8}$$
$$K_{FLmin2} = -(O_{FL} - \sqrt{\mu_{FL}}/2) + 20 \tag{9}$$

$$K_{HLmax1} = -(O_{HL} + \sqrt{\mu_{HL}}/2) + 40 \tag{10}$$
$$K_{HLmax2} = -(O_{HL} - \sqrt{\mu_{HL}}/2) + 40 \tag{11}$$
$$K_{HLmin1} = -(O_{HL} + \sqrt{\mu_{HL}}/2) - 5 \tag{12}$$
$$K_{HLmin2} = (O_{HL} - \sqrt{\mu_{HL}}/2) - 5 \tag{13}$$

where O_{FL} and O_{HL} are the offsets of the fore and hind hip joints, respectively. Finally, knee angles are given by $[max(min1, min2)min(max1, max2)]$.

Table 1. Parameter Limits

Parameters	Lower	Upper
μ_{FL}	0.0001	3600
$O_{FL}(^\circ)$	$-1600 + \mu_{FL}/2$	$400 - \mu_{FL}/2$
μ_{HL}	0.0001	3600
$O_{HL}(^\circ)$	$-400 + \mu_{FL}/2$	$1600 - \mu_{HL}/2$
ω_{sw}(rad/s)	1	12
$K_{FL}(^\circ)$	$max(K_{FLmin1}, K_{FLmin2})$	$min(K_{FLmax1}, K_{FLmax2})$
$K_{HL}(^\circ)$	$max(K_{FLmin1}, K_{FLmin2})$	$min(K_{HLmax1}, K_{HLmax2})$

3.3 Genetic Algorithm

Genetic Algorithms (GA) start from a pool of points, usually referred to as chromosomes. Chromosomes represent potential optimal solutions of the problem being solved. In order to implement a GA, it is necessary to define the representation of the search space and a fitness function which permits the comparison between the different chromosomes. Furthermore, genetic operators and the selection mechanism must also be defined.

One or several optimal combinations of amplitude and offset for the hip oscillators, offset for the knees and swing frequency are necessary in order to generate the desired forward locomotion movement, as explained before. Each chromosome consists in 7 CPG free parameters, as shown in fig. 3, that span our vector space for the optimization.

Chromosome

μ_{FL}	O_{FL}	μ_{HL}	O_{HL}	ω_{sw}	K_{FL}	K_{HL}

Figure 3. A chromosome is made of seven free parameters.

In our optimization system, we begin the GA search by randomly generating an initial population of chromosomes.

The GA selection operator assures that chromosomes are copied to the next generation with a probability associated to their fitness values. Therefore, this operator mimics the survival of the fittest in the natural world. Although selection assures that in the next generation the best chromosomes will be present with a higher probability, it does not search the space, because it just copies the previous chromosomes. The search results from the creation of new chromosomes from old ones by the application of genetic operators. The crossover operator, takes two randomly selected chromosomes; one point along their common length is randomly selected, and the characters of the two parent strings are swapped, thus generating two new chromosomes. The mutation operator, randomly selects a position in the chromosome and, with a given probability, changes the corresponding value. This operator does assure that new parts of the search space are explored, which selection and crossover could not fully guarantee.

In this work, real representation of the variables was considered. So, each vector consists of a vector of real values representing the decision variables of the problem. Genetic operators were chosen taking into account this representation. In order to recombine and mutate chromosomes, the Simulated Binary Crossover (SBX) and Polynomial Mutation were considered, respectively. These operators simulate the working of the traditional binary operators [3]. In order to select chromosomes for the application of genetic operators, a tournament selection was implemented.

3.4 Fitness Specification

The performance of each chromosome is evaluated according to the robot body vibration (f_a), the forward velocity (v) and the Wide Stability Margin WSM.

3.4.1 Vibration

We consider that a good gait should have less vibration, because the robot is subjected to less strain. In order to calculate the total vibration we sum the standard deviation of the measures of the (a_x, a_y, a_z) accelerometers built-in onto the robot, similarly to [19, 6, 20], as follows:

$$f_a = std(a_x) + std(a_y) + std(a_z) \tag{14}$$

3.4.2 Wide Stability Margin

For stability, we calculate the wide stability margin [21] (WSM). This is a measure of the locomotion stability that provides the shortest distance between the projection of the center of mass in the ground and the polygon formed by the vertical projection in the ground of robot feet contact points. A gait is considered better when has a higher WSM.

3.4.3 Velocity

We calculate the forward velocity using the traveled distance of the robot during the evaluation of each chromosome of the population, *i.e.* during 12 seconds. A gait is considered better if it achieves higher velocities.

We intend to determine the best gait considering minimization of the body vibration and maximization of the velocity and wide stability margin. The normalize fitness is given by:

$$\text{fitness}_{total} = W_{\text{a}} * \frac{f_a}{f_{\text{a,max}}} + W_v * \frac{1}{v} * v_{\min} + W_{\text{wsm}} * e^{-\frac{wsm}{wsm_{\max}}}, \tag{15}$$

where W_a,W_v and W_{wsm} are the vibration, velocity and WSM weights, respectively.

For each fitness component to have the same significance, we normalize the values of the fitness components. We have determined that $v_{\min} = 10(mm/s)$, $f_{\text{a,max}} = 0.4$ and $wsm_{\max} = 65$. These are the maximum values that the fitness components may achieve.

3.5 Weights of the fitness

We apply weights to each component of the fitness function, similarly to [20]. We have three weights, one for each fitness component: W_{a} (vibration weight), W_{v} (velocity weight) and W_{wsm} (WSM weight). The sum of the three components is always 1, as follows:

$$W_{\text{a}} + W_{\text{v}} + W_{\text{wsm}} = 1. \tag{16}$$

We implemented a method for the computation of the weights such that the component weights change depending on the value of the components. Lower velocities give higher weights for the velocity component but higher velocities give lower weights for the corresponding component. This method is shown in fig 4. It is based on a negative exponential, such that the higher the velocity, the lower the velocity weight (W_v), as follows:

$$W_{\text{v}} = 0.7 \times \exp^{-(v \times 0.01)} \tag{17}$$

We want to minimize the overall vibration but to maximize the velocity. Then, we want lower vibrations for high velocities. This is achieved by setting

$$W_{\text{a}} = 0.7 - W_{\text{v}} \tag{18}$$

Finally,

$$W_{\text{wsm}} = 0.3 \tag{19}$$

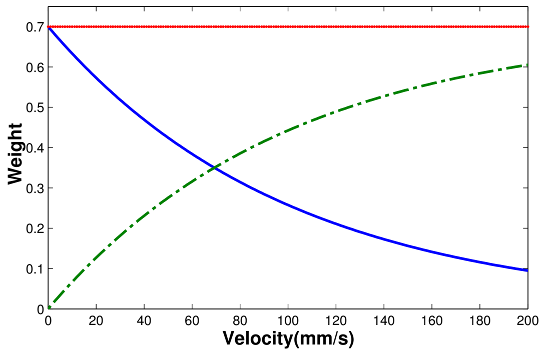


Figure 4. Weights W_v (solid line) and W_a (dashed line) for different velocity values. The sum between W_v and W_a is the dotted line.

4 SIMULATION RESULTS

In this section, we describe the experiment done in a simulated ers-7 AIBO robot using Webots [17]. The working scneraio is shown in fig 5. Webots is a software for the physic simulation of robots based on ODE, an open source physics engine for simulating 3D rigid body dynamics. The model of the AIBO is as close to the real robot as the simulation enable us to be. We simulate the exact number of DOFs and mass distributions.

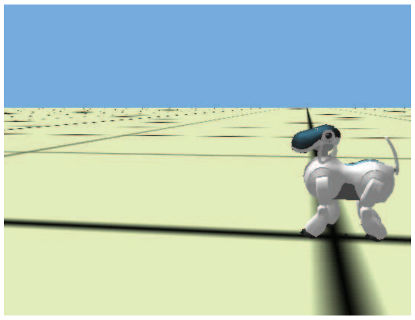


Figure 5. Simulation experimental setup.

The ers-7 AIBO dog robot is a 18 DOFs quadruped robot made by Sony. The locomotion controller generates trajectories for the joint angles of the hip and knee joints in the sagittal plane, that is 8 DOFs of the robot, 2 DOFs in each leg.

At each sensorial cycle (30 ms), sensory information is acquired.Each chromosome is evaluated during 12 seconds. We apply the Euler method with 1ms fixed integration step, to integrate the system of equations. At the end of each chromosome evaluation the robot is set to its initial position and rotation.

In our implementation, the optimization system ends when the number of generations exceeds 50 generations. We depict results when a population was established with 50 chromosomes and a preset number of 50 generations was set.

The generated gaits have a fixed duty factor $\beta = 0.75$ and a relative phase $\varphi_{\mathrm{LH}} = 0.75$.

Table 2 contains the Best, Mean and standard deviation (SD) values of the solutions found (in terms of fitness function and time) over 10 runs.

Table 2. Performance of GA algorithm in the optimization system

Fitness			Time (hours)		
Best	Mean	SD	Best	Mean	SD
0.2727	0.2973	0.0212	3.5350	4.0843	2.0975

Fig. 6 shows the evolution of all the 10 runs (lighter lines), best (solid line) and mean (dashed line) fitness function value over 50 generations. The best individual has a fitness value of 0.2727 that was achieved at generation 50. The best run took 3h53 min (CPU time) and each generation took in average 294.07 seconds.

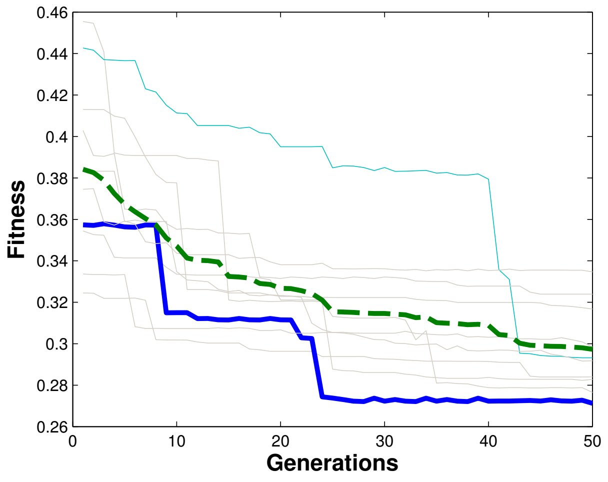


Figure 6. Fitness evolution during 50 generations.

Table 3, shows the parameters values of the best chromosome of the first and last generation.

Table 3. Optimization Parameters Results

Parameters	**1st Generation**	**50th Generation**
μ_{FL}	2692.6	66.26
$O_{FL}(^\circ)$	-12.77	8.19
μ_{HL}	1954.5	252.81
$O_{HL}(^\circ)$	9.21	15.87
$\omega_{sw}(rad/s)$	11.87	10.62
$K_{FL}(^\circ)$	81.64	52.65
$K_{HL}(^\circ)$	30.34	5.00
Fitness	0.3573	0.2727

Fig. 7 depicts the evolution of the measurements of sensor data, vibration, velocity and WSM of the best chromosome of each generation.

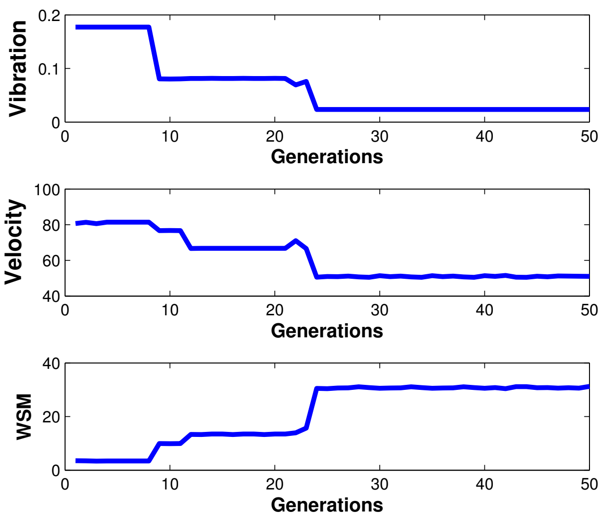


Figure 7. Evolution of the sensor measurements during the 50 Generations.

Table 4 lists the values of the vibration, velocity and WSM for the first and last generation. Velocity is $0.08623(ms^-1)$ and $0.05098(ms^-1)$ for the first and 50th generation, respectively. These values may seem much worst then those achieved by previous works, specially in the RobotCup domain. However, the robot configuration was the required to achieve higher velocities: the robot knees were completely folded. In our work, we just try to achieve a higher velocity for a crawl gait. In fact, it is a slow gait since three legs are kept in ground contact. But gait specification, duty factor and relative phase, were maintained as expected.

Table 4. Optimization Sensor Results

Generation	$Vibration$	$Velocity(mm/s)$	$wsm(mm)$	**Fitness**
1st Generation	0.177	80.623	3.508	0.357
50th Generation	0.0233	50.980020	31.253	0.271

5 Conclusion

In this article, we have addressed the locomotion optimization of a quadruped robot that walks with a walking gait.

A locomotion controller based on dynamical systems, CPGs, generates quadruped locomotion. These CPG parameters are tuned by an optimization system. This optimization system combines CPGs and a genetic algorithm. As a result, sets of parameters obtained by the genetic algorithm were adequate for the implementation of a locomotion walking gait with a velocity of 50.98 (mm/s), low vibration and a high wide stability margin.

Currently, we are using other optimization methods such as evolutionary strategies and electromagnetism algorithm. We will extend this optimization work to address other locomotion related problems, such as: the generation and switch among different gaits according to the sensorial information and the control of locomotion direction.

We further plan to extend our current work to implementation on the Aibo ers7 the locomotion optimization similarly to [2].

6 Acknowledgments

Work supported by the Portuguese Science Foundation (grant PTDC/EEA-CRO/100655/2008).

REFERENCES

[1] Luiz Castro, Cristina Santos, Miguel Oliveira, and Auke Ijspeert, 'Postural control on a quadruped robot using lateral tilt: A dynamical system approach.', in *EUROS*, volume 44 of *Springer Tracts in Advanced Robotics*, pp. 205–214. Springer, (2008).

[2] Sonia Chernova and Manuela Veloso, 'An evolutionary approach to gait learning for four-legged robots', in *In Proceedings of IROS'04*, (September 2004).

[3] R.B. Deb, K. Agrawal, 'Simulated binary crossover for continuous search space.', *Complex Systems*, **9(2)**, 115–149, (1995).

[4] Sarah Degallier, Cristina Santos, Ludovic Righetti, and Auke Ijspeert, 'Movement generation using dynamical systems: a humanoid robot performing a drumming task', in *IEEE-RAS International Conference on Humanoid Robots*, (2006).

[5] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[6] Dragos Golubovic and Huosheng Hu, 'Evolving locomotion gaits for quadruped walking robots', *Industrial Robot: An International Journal*, **32**, 259 – 267, (2005).

[7] S. Grillner, 'Locomotion in vertebrates: central mechanisms and reflex interaction.', *Physiological Reviews*, **55**, 247–304, (1975).

[8] S. Grillner, 'Neurobiological bases of rhythmic motor acts in vertebrates', *Science*, **228**(4696), 143–149, (1985).

[9] Freyr Hardarson, *Stability analysis and synthesis of statically balanced walking for quadruped robots*, Ph.D. dissertation, KTH, 2002.

[10] Gregory S. Hornby, Seichi Takamura, Takashi Yamamoto, and Masahiro Fujita, 'Autonomous evolution of dynamic gaits with two quadruped robots', *IEEE Transactions on Robotics*, **21**, 402–410, (2005).

[11] Gia Loc Vo Tran Duc Trong Young Kuk Song Ig Mo Koo, Tae Hun Kang and Hyouk Ryeol Choi, 'Biologically inspired control of quadruped walking robot', *International Journal of Control, Automation and Systems*, **7**(4), 577–584, (2009).

[12] K. Inagaki and H. Kobayashi, 'A gait transition for quadruped walking machine', *Proceeding of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Ssytems*, (1993).

[13] Min Sub Kim and William Uther, 'Automatic gait optimisation for quadruped robots', in *In Australasian Conference on Robotics and Automation*, (2003).

[14] Nate Kohl and Peter Stone, 'Machine learning for fast quadrupedal locomotion', in *in The Nineteenth National Conference on Artificial Intelligence*, pp. 611–616, (2004).

[15] Vítor Matos, Cristina P. Santos, and Carla M. A. Pinto, 'A brainstem-like modulation approach for gait transition in a quadruped robot', in *IROS*, pp. 2665–2670, (2009).

[16] R. B. Mcghee and A. A. Frank, 'On the stability properties of quadruped creeping gaits', *Mathematical Biosciences*, **3**(1-2), 331–351, (August 1968).

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

[18] Ludovic Righetti and Auke Jan Ijspeert, 'Pattern generators with sensory feedback for the control of quadruped locomotion', *IEEE International Conference on Robotics and Automation*, (2008).

[19] Thomas Röfer, 'Evolutionary gait-optimization using a fitness function based on proprioception', in *RobuCup*, pp. 310–322, (2004).

[20] Manish Saggar, Nate Kohl, and Peter Stone, 'Autonomous learning of stable quadruped locomotion', in *RoboCup2006: Robot Soccer World Cup X*. Springer Verlag, (2007).

[21] S. Song and K. Waldron, *Machines that Walk: The Adaptive Suspension Vehicle*, MIT Press, Cambridge, 1989.

[22] Kobayashi Toshiya Inoura Takashi Masuda Tatsuya Tsujita, Katsuyoshi, 'A study on adaptive gait transition of quadruped locomotion', in *none*, pp. 2489 – 2494, Tokyo, 113, Japan, (2008).

Evaluating Reinforcement Learning Methods for Robot Navigation in Home Environments

Ruben Gerlach

Abstract. Developing navigation systems for robots that are effective while preserving the safety of near inhabitants is one of the hard challenges modern robot industry is facing. In this paper we present a promising approach for training such strategies using Reinforcement Learning architectures that are independent of the maps they are trained on or the number of inhabitants around. For this purpose we use state extrapolation to encode a state signal that presents the essence of a given situation without losing any vital information.

1 Introduction

In our present world robots gain ongoing popularity in industry and home environments. Be it as autonomous vacuum cleaners or mechanical mans's best friend, especially commercial marketing is increasingly being focused on. At the same time, navigation and respect to the inhabitants leave room for improvement. In the described cases, collisions with people or interior may be harmless, yet as the range of tasks for robots increases, a growth of mass and size is to be expected. Solutions need to be found that guarantee the safety of the users while the underlaying navigation becomes as efficient as possible to allow precise and quick accomplishment of tasks. In our study ([4]) we develop several concepts that address robot navigation in home environments with the respect to the appearing inhabitants using flat and hierarchical reinforcement learning and compare them in a realistic simulation setting. We show that reinforcement learning is able to generate promising strategies that are much more efficient and safe than the hand coded solutions we created for comparison. The results of this study are presented in this paper.

2 Reinforcement Learning Theory

Reinforcement Learning (RL) is a popular method to let an agent learn problem solving strategies in unknown and complex domains. In each state s_t the agent takes an action a_t, where t denotes a discrete time step. The action is performed in the environment and the agent receives a resulting reward r_{t+1} that denotes how well the agent reacted, as well as the resulting state s_{t+1}. The given rewards are defined by the user to achieve a certain goal, e.g. in our case negative rewards (i.e. punishments) would be given when the agent causes a collision with an inhabitant so the agent would avoid these situations in the future. To maximize the total amount of rewards received during an episode or in continuous tasks, a state-action value function is maintained, that maps from state-action pairs to expected accumulated rewards following the current strategy. Given the true, or optimal state-action value function, one would only have to choose the action that returns the highest value in the current state and the agent would follow the optimal strategy in the means of the reward distribution. Yet as the value function is unknown at the beginning of the training, it has to be updated according to the experience of the agent and with a certain probability exploratory actions have to be taken in contrast to best rated actions adjust actions that are under- or overrated. The successful application of RL methods in the field of agent navigation and similiar cases has been shown widely. RL methods have been applied for the popular robot soccer leagues known as Robocup, see for example [13] and [9], in First-Person-Shooters, see [5], strategy games unit navigation (presented in [7]) and many more. There exist several algorithms to update value functions in RL tasks that perform well in practise, see [14] for a detailed description. In our study we have implemented and compared the most popular algorithms such at Sarsa, Q-Learning, different Monte Carlo methods and more. The most successful in our case have been proven to be the Sarsa(λ) algorithm from [10], see [6] for an extensive analysis, combined with replacing traces (compare [12]), as well as the Hierarchical Semi-Markov Q-Learning (HSMQ) algorithm. HSMQ works in principle like Sarsa(λ), except that there exist several hierarchy layers where each layer may define non atomic subtasks that can be chosen as if they were common actions. These subtasks call a lower layer in the hierarchy that is executed until a state is reached that is defined terminal for this subtask. When a subtask is finished the agent will return to the previously active layer using the sum of rewards that have been gained during the execution of the subtask as reward for the chosen subtask as if they were common action and reward (see [3]). For function approximation we use Tile Coding, see [11] for a brief overview, and neural networks using the Resilient Propagation (RPROP) update algorithm, see [8] or [1].

3 The Testbed

As RL tasks require up to millions of repetitions to acquire successful strategies, using a real robot in a real environment would take unsatisfying long. For this purpose we made initial tests with the popular Player/Stage framework (see [2]) to be used as a testbed that also would have been fully compatible to our robot that uses the Player framework as well. We found out that Player/Stage uses a time critical update function and hence experiences great simulation differences when not being run in real time, so even if a simulation, not much time would have been won in our case. For this reason we decided to implement our own time step independent simulation testbed.

3.1 Tested Maps

We used three different maps to train and test our agents on. The main map we used is shown in Fig. 1a). In this the different agents have been trained and tested, while we used Fig. 1b) and Fig. 2 to verify the trained strategies. The maps Simplerooms and Section are both taken from [2]. The waypoints for the agents and inhabitants have been set by hand such that they are equally distributed over the maps.

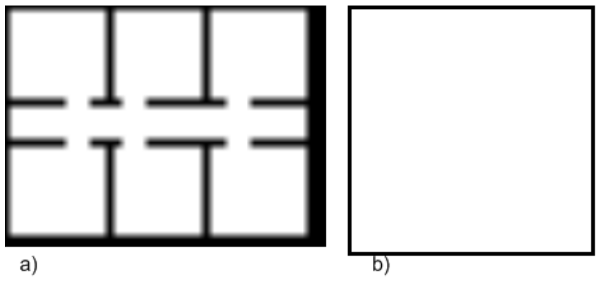


Figure 1. Maps: a) Simplerooms (from [2], reworked) b) Arena.

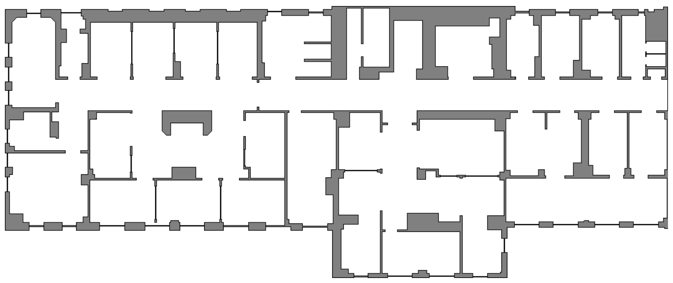


Figure 2. Map: Section (from [2], reworked).

3.2 The Inhabitants

In our testbed we are able to add as many inhabitants as needed with individual waypoints for each of them. The inhabitants choose randomly among a set of given waypoints, run up to them and then repeat. If at any time the agent is in the way, the inhabitant stops and waits for the agent to clear the path. There were other possibilites for our inhabitants to react if an agent crosses the way, such as ignoring the agent and move on, risking a collision, or even evade the situation themselves. We took the described behavior because its close to the reaction of a real human without making it the agent too easy to escape the situation.

3.3 Handcoded Solutions

For comparison reasons we wrote different hand coded solutions to compare the RL strategies with, namely **Random**, **Planner** and **Evade**. The **Random** agent decides randomly between following the planner to move to the current waypoint or escape into one of four possible directions. This behavior can be used to compare if an RL agent is able to improve the behavior at all. The **Planner** agent moves up to the waypoints all time ignoring any inhabitants. This agent allows to estimate the probability of collisions and the theoretical maximum speed for reaching waypoints under the given circumstancens (i.e. the current map and number of inhabitants). The last agent uses the extrapolation unit that is also used within our RL agents: From the current state, the outcome of the possible evade actions (moving into one of four directions) is estimated, as well as the actions of the inhabitants regarding their current velocity and orientation. For each of these extrapolated states, we measure the distance between the agent and the closest inhabitant leading to four different values. A fifth value is generated using the current position of the agent, which we use to represent the outcome of the planner unit. We chose not to take into account the extrapolated position after following the planner as we would like to be independent of specific implementation details. The **Evade** agent decides upon these five values which action to take: As long as the planner distance is greater than 3 meters, this action is taken, else the evade action leading to the greatest distance to the inhabitants is chosen. The distance for the planner action are variable and 3 meters have been indicated to be the most successful value by previous testruns.

3.4 Measurement of Success

Finally we measure different categories to compare the hand coded solutions and the various RL agents: First of all, we take into account the ratio of unsuccessful episodes (i.e. episodes that end in a collision) to all episodes, which is the main indicator for how successful the agent is. Categories of second order are the average speed per waypoint and the average blocking of inhabitants by the agent.

4 The Setup

In the software that was developed for our study, we are able to combine all implemented state representations, action macros, RL algorithms (such as Sarsa(λ) or HSMQ), value function approximators (we used tile coding and neural networks), rewarding systems and RL exploration strategies individually. The implemented candidates for each category are described in the following sections:

4.1 State Encoders

The state encoder is the unit that creates a multidimensional state vector of the observed environment to be used by the RL algorithm. The values of the state vector are scaled to [0, 1] for tile coding and [-1, 1] for the use of neural networks.

4.1.1 AllRelative

For every single inhabitant the following values are encoded:

- Angle from agent to inhabitant
- Angle from inhabitant to angle
- Distance between agent and inhabitant
- Velocity of the inhabitant
- Velocity of the agent

4.1.2 AllRelativeNoSpeeds

Like *AllRelative*, omitting the velocities.

4.1.3 AbsoluteAngles

Like *AllRelativeNoSpeeds*, this time using absolute angles, i.e. the angles now have values in [0, π] (instead of [$-\pi$, π] before).

4.1.4 *AreaBased*

The map the agent is trained on is manually divided into reasonable areas. For the agent and each inhabitant, the id of the current area is encoded. This might enable the agent to learn when to omit specific areas because they are currently frequented by the inhabitants. Fig. 3 shows the 43 different areas for the map Simplerooms we created. Note that in this case the complexity already is $43^3 = 79.507$ for two inhabitants and the agent, not including different actions, let alone the possibility of bigger maps.

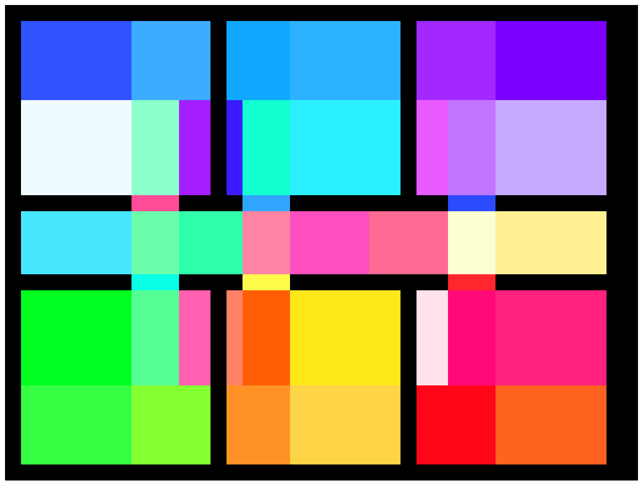


Figure 3. Simplerooms manually divided into 43 areas.

4.1.5 *ClockDescr*

For each inhabitant, the distance and the relative position to the agent are encoded, see fig. 4. In addition, the distance from the agent to the closest wall in every direction is written, to allow safe navigation in the environment.

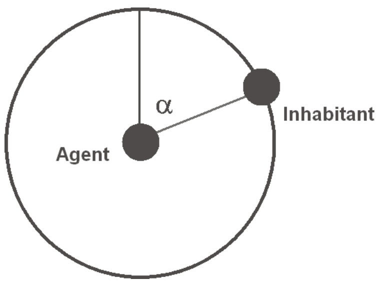


Figure 4. Relative position of an inhabitant to the agent, labeled by the angle α.

4.1.6 *OrthogonalDescr*

Like *ClockDescr*, though instead of distance and angle the relative position is encoded by the distance vector, i.e. its coordinates x and y.

4.1.7 *ClockDescrWithOrientations*

Like *ClockDescr*, this time adding the orientation of every inhabitant.

4.1.8 *ClockDescrWithPosId*

Like *ClockDescr*, this time encoding the x and y coordinates of the agent instead of the distances to the walls, to allow learning the specific properties of each part of the map.

4.1.9 *MultiCake*

The previously described state spaces all have one severe drawback: Their complexity increases with the number of inhabitants on the map. To fight this issue, the state encoder *MultiCake* was created: For each of the four directions, the distance to the *closest* inhabitant is remembered, hence reducing the state signal by dropping inhabitants that are further away or forming a group.

4.1.10 *MultiCakeWithOrientations*

Like *MultiCake*, but removing all inhabitants that face away from the agent and hence may not have a significance to the current situation.

4.1.11 *MinimalDistancePredictor*

For each of the actions Stop and Move into one of four directions, the current situation is extrapolated for some frames, i.e. every inhabitant is moved in respect to its current velocity, as well as the agent by the velocity of the chosen action. After the extrapolation, the closest distance from the agent to any inhabitant is measured and remembered. This leads to five different values for each of the named actions that represent the state signal.

4.2 Action Sets

The action set of an agent defines its options to interact with the environments. Depending on the given state encoder, we try to choose an action set that integrates with the described features of the environment. E.g. it makes no sense to use an action set that defines actions for moving into one of the four directions up, down, left and right when the state encoding only describes angles and positions relative the orientation of the agent. The action sets we have tested are described in the following sections.

BackwardOrPlanner: A simple action set that defines the two actions *Follow the Planner* and *Move Backwards*.

BackwardForwardOrPlanner: Like *BackwardOrPlanner* but adding the third action *Move Forward*.

PlannerOrCrossNav: Defines movement actions to move up, down, left and right or following the planner.

PlannerOrRoseNav: Like *PlannerOrCrossNav* but adding additional movement directions between the four given, resulting in actions for evading into eight different directions plus *Follow the Planner*.

PlannerOrNonAtomicCrossNav: Like *PlannerOrCrossNav*, only this time instead of moving into one of the four directions, a target point some distance away is given to which the agent moves. The action is hence non atomic but enduring several frames of simulation. If a wall is closer than the target point, the target point is brought forward.

PlannerOrNonAtomicCrossNavAreaBased: Invented for the state encoder *AreaBased*: Like *PlannerOrCrossNav*, but the agent will move into the chosen direction until the state signal changes to prevent great variances due to randomness in the learning process while the state (the occupied areas) remains unchanged.

HierarchicalEvading: To allow the use of hierarchical RL architectures, this action set features the actions *Follow the Planner* and *RLEvade*, which itself points to a secondary RL hierarchy that may use any configuration to achieve evading from the inhabitants.

CrossNav: Created to be used in the lower level of a hierarchical setup that learns to evade the inhabitants, this action set only defines actions for moving into one of the four directions up, down, left and right.

4.3 RL Algorithms

Allthough many papers suggest Sarsa(λ) to be a very successful algorithm, we wanted to measure the performance of different RL algorithms to assure the best results. These are the algorithms we implemented and compared:

- One-Step Q-Learning
- One-Step Sarsa
- First-Visit MC
- Off-Policy First-Visit MC
- Sarsa(λ)
- Watkins' Q(λ)
- Peng's Q(λ)
- Naive Q(λ)
- HSMQ

See [14] for details on the algorithms.

4.4 Rewards

We experimented with different events and values for the rewarding systems. Only the two final configurations are presented here. Table 1 shows the events for the flat RL architectures, the events for the secondary layer of the hierarchical architecture are given in Table 2. The first layer, that decides between *Follow the Planner* and *RLEvade* basically receives the same rewards as the flat architecture except for the reward of the evade action which is the accumulated rewards that was generated by the evade layer. More information about the hierarchical organization of the navigation task will be given later.

Event	Reward
Collision	-1
Blocking one or more inhabitants	-0.05
Evade action	-0.0001
Following the planner	0

Table 1. Rewards in the flat RL architecture.

5 Results

In the initial testruns, we trained reasonable combinations of the previously described state encodings and action spaces using Sarsa(λ) on the map Simplerooms with one inhabitant over $50,000$ episodes. The results of these testruns are shown in Table 3.

Event	Reward
Collision	-1
Blocking one or more inhabitants	-0.01
else	-0.0001

Table 2. Rewards in the second hierarchy layer of the hierarchical architecture.

State Encoder	Action Set	Collisions (%)	Duration	Blocking (%)
AllRelative	*BackwardOrPlanner*	33.50	202.27	3.24
AllRelative	*BackwardForwardOrPlanner*	29.60	250.02	3.86
AllRelativeNoSpeeds	*BackwardForwardOrPlanner*	34.50	319.80	4.97
AbsoluteAngles	*BackwardForwardOrPlanner*	35.60	224.58	4.47
AreaBased	*PlannerOrCrossNav*	4.20	601.82	0.15
AreaBased	*PlannerOrNonAtomicCrossNavAreaBased*	1.40	447580	0.12
ClockDescr	*PlannerOrCrossNav*	0.00	423.30	0.00
ClockDescr	*PlannerOrNonAtomicCrossNav*	1.10	2481800	0.01
ClockDescrWithOrientations	*PlannerOrCrossNav*	0.20	461.27	0.00
ClockDescrWithPosId	*PlannerOrCrossNav*	2.30	495.80	0.15
MultiCake	*PlannerOrCrossNav*	4.30	464.96	0.22
MultiCakeWithOrientations	*PlannerOrCrossNav*	0.00	∞	0.00
MinimalDistancePredictor	*PlannerOrCrossNav*	0.10	329.61	0.01
MinimalDistancePredictor	*PlannerOrRoseNav*	1.80	325.67	0.16
MinimalDistancePredictor	*PlannerOrNonAtomicCrossNav*	85.80	20160	2.13

Table 3. Comparison of different state encodings and action spaces on the map Simplerooms with one inhabitant. The agent use Sarsa(λ) with $\gamma = 0.8$ and $\lambda = 0.7$, Tile Coding with one Tiling and static exploration with $\epsilon = 0.05$.

It was early observable that the state encoder *AllRelative* in all its variants was unable to let the agent learn efficient evading behavior. Because no information about walls are encoded, the collision rate on the map Simplerooms is higher than it would be on the map Arena. The state encoder *ClockDescr* is able to achieve a collision rate of 0% within the first $50,000$ episodes, drawback of course is the growing complexity depending on the number of inhabitants encoded. Even with only two inhabitants the training process takes much longer and converges against a significantly higher failure ratio. The variant *ClockDescrWithPosId*, that encodes the absolute position of the agent instead of the distances to the surrounding walls, achieves a slightly higher efficiency in terms of speed, in this case the trained strategy depends on the map it was trained on, of course. The state encoder *AreaBased* shows similiar behavior: A clear learning success can be noted, the complexity though again depends on the number of inhabitants and the learned strategy is also only valid on one map. The state encoder *MultiCake* reaches good performance and is in addition independent of the number number of inhabitants. The variant *MultiCakeWithOrientations* however, in which inhabitants that face away from the agent are ignored, is not able to produce any useful strategy at all. The agent only executes evading actions and does not follow the planner anymore. *MultiCake* is only outperformed by *MinimalDistancePredictor* which achieves a notable collision rate of 0.1% within the first $50,000$ epsiodes and is also independent of the number of considered inhabitants. Moreover no other state encoder is able to produce such short pathes. From the actions sets *PlannerOrCrossNav* is the only one that produces convincing results, any other one performs clearly worse. Altogether the combination of *MinimalDistancePredictor* and *PlannerOrCrossNav* seems promising and fast. In supplemental trainings the failure ratio could be improved and the advantage over the other candidates could be emphasized. In the following sections this configuration is focused and the results of further optimizations are presented.

5.1 Improving the performance

Comparable papers suggest hierarchical architectures to be a very powerful way to increase the performance of RL applications as they drastically reduce the complexity by distributing decisions over several RL layers. In our study we developed a HSMQ architecture that divides the navigation task into two RL layers: The first decides whether to follow the planner or to take an evade action, the second layer represents the evade action by using the action set *CrossNav*. Episodes are ended whenever the agent reaches a distance of at least 2 to each inhabitant or a collision occurs. To maximize the performance of the agents we have in addition carefully explored the effects of tuning the different parameters, applying the different RL algorithms and changing the function approximations on the flat RL architecture and both layers of the hierarchical architecture. The influence of these variations shall exemplarily be discussed for the evade layer of the HSMQ architecture.

5.1.1 Tuning γ

To start off we began tuning the γ parameter using the popular Sarsa(λ) algorithm with $\lambda = 0.7$ and Tile Coding with one tiling for function approximation. It was to be expected that the agent would perform better with higher values, as it would allow to forsee the outcome of the relatively short episodes in the evade layer. Fig. 5 shows the performance for different γ values over the first $27,000$ episodes. Interestingly, $\gamma = 1$ has a very high rate of failure until it suddenly drops down. $\gamma = 0$ on the other hand learns surprisingly well. Further investigations suggest that this is due to the large generalization we used in the function approximation in this run. It is obvious that these results are only valid for the evading layer as its episodes are very short compared to the other tasks we defined.

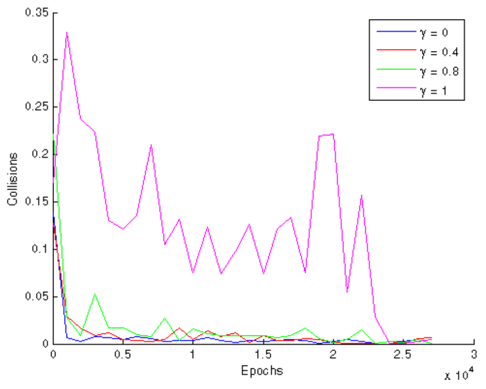


Figure 5. Comparison of different γ values on the map Simplerooms with two inhabitants using Sarsa(λ) with $\lambda = 0.7$, Tile Coding with $\alpha = 0.1$, one tiling and six tiles per dimension, and a static exploration with $\epsilon = 0.05$.

5.1.2 Applying Artificial Neural Networks

Using a finer grain of approximation generalization the outcome looks more as expected. Fig. 6 shows the same configuration as before, this time using artificial neural networks with 160 neurons in the hidden layer for approximation. Besides the more conventional graphs of the performance, the final collision rate is noteably higher which indicates that the neural networks need more time to learn the strategies as well as several runs are needed to train nets that can compete with a tile coding solution.

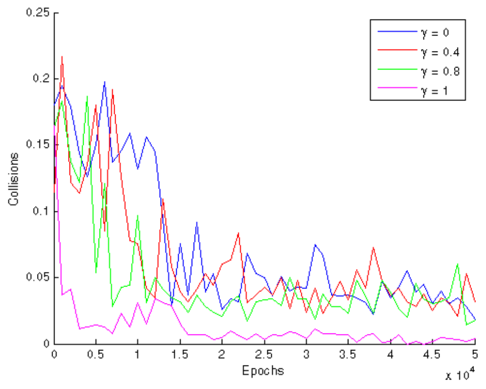


Figure 6. Comparison of different γ values on the map Simplerooms with two inhabitants using Sarsa(λ) with $\lambda = 0.7$, articifial neural networks with the RPROP update algorithm and a static exploration with $\epsilon = 0.05$.

5.1.3 Tuning λ

Fig. 7 displays the results of testing different λ values. This time the configurations perform as suggested by literature and comparable works, $\lambda = 0.7$ shows the fastest learning behavior but the other values do not work too bad either. Only $\lambda = 1$ performs remarkably worse which already shows the similarity to the Monte Carlo algorithms that are tested in the next section.

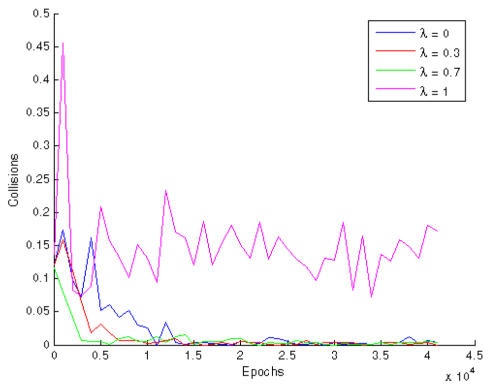


Figure 7. Comparison of different λ values on the map Simplerooms with two inhabitants using Sarsa(λ) with $\gamma = 1$, Tile Coding with $\alpha = 0.1$, one tiling and six tiles per dimension, and a static exploration with $\epsilon = 0.05$.

5.1.4 Comparison of different RL algorithms

Now with the use of the previously evaluated parameters we trained compared the different RL algorithms listed before. Fig. 8 shows the results for two Monte Carlo variants, namely First-Visit MC and Off-Policy MC, as well as the classic algorithms Sarsa and Q-Learning. Fig. 9 compares the tested Eligibility Traces algorithms:

Sarsa(λ), Naive Q(λ), Watkins Q(λ) and Pengs Q(λ). Details about these algorithms are given in [14], we used replacing traces in all cases. Sarsa(λ) clearly shows the fastest convergence, Sarsa and Q-Learning both produce a good performance as well. The Monte Carlo variants perform suriprisingly weak but have shown convincing behavior when applied not to the evading controller but other tasks, even if in no case as well as Sarsa(λ).

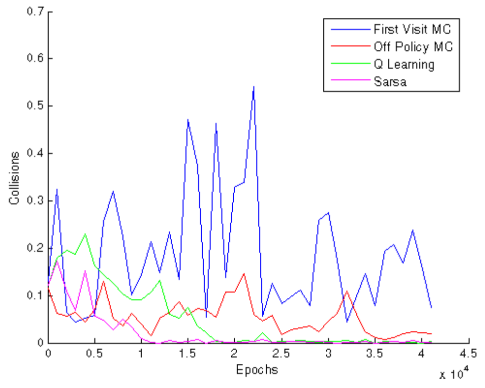


Figure 8. Comparison of different RL algorithms on the map Simplerooms with two inhabitants with $\gamma = 1$, Tile Coding with $\alpha = 0.1$, one tiling and six tiles per dimension, and a static exploration with $\epsilon = 0.05$

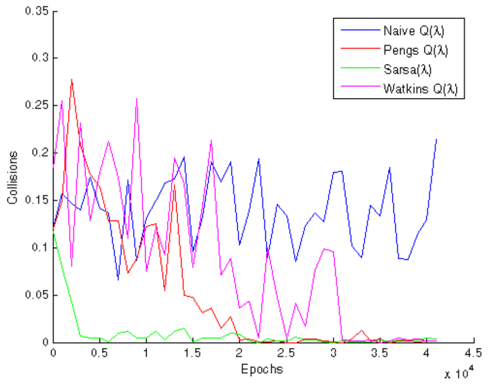


Figure 9. Comparison of different RL algorithms on the map Simplerooms with two inhabitants with $\lambda = 0.7$, $\gamma = 1$, Tile Coding with $\alpha = 0.1$, one tiling and six tiles per dimension, and a static exploration with $\epsilon = 0.05$

5.2 Final Results

Map	Agent	Collisions (%)	Duration	Blocking (%)
Simplerooms, 2 Inhabitants	Flat RL	0.42	477.09	3.84
	HSMQ	0.01	437.39	3.58
	Evade	6.41	481.89	8.58
	Planner	52.14	216.96	8.78
	Random	99.72	2,003.8	10.72

Table 4. Results of the RL agents on Simplerooms compared to the handcoded solutions.

In this section we present the results of our approach both for the flat RL and the hierarchical RL architectures. The final configurations we found look as follows:

Flat RL: Sarsa(λ) with $\lambda = 0.7$, state encoding: *MinimalDistancePredictor*, action set: *PlannerOrCrossNav*, the rewards represented in Table 1, an ϵ-Greedy policy with $\epsilon = 0.05$ and an individual neural network with 160 neurons in the hidden layer for each of the five possible actions using the RPROP update algorithm.

Hierarchical RL: HSMQ with $\lambda = 0.7$, state encoding: *MinimalDistancePredictor* for each hierarchy layer, a decision between following the planner and a reference to the second layer in the first hierarchy layer, and the action set *CrossNav* in the second hierarchy layer, hence representing the evade task. The second layer was executed until a distance of at least 2 meters was reached to each of the inhabitants (or a collision occured and the episode was ended anyway). The second layer received rewards as described in Table 2, while the first layer was rewarded as described in Table 1, except that this time the reward for the evade action was not -0.0001 but the accumulated reward of the second layer, just as the HSMQ algorithm suggests. Furthermore an ϵ-Greedy policy with $\epsilon = 0.05$ and an individual neural network with 160 neurons in the hidden layer for each of the six possible actions using the RPROP update algorithm.

Each configuration was trained several times until convergence occured and the best results were taken for comparison with the handcoded solutions. At first, we show the results of the two configurations on the map they have been trained on. Fig. 10 shows the graphs for the collision rate of the HSMQ architecture where datapoints have been measured every $10,000$ episodes using the average outcome of $1,000$ episodes with exploration disabled. The secondary layer, responsible for choosing the direction to which to evade, was pretrained in a separate run. Graphs for the duration and blocking rate are shown in Fig. 11 and Fig. 12. The graphs show that HSMQ very fastly generates a strategy that performs well and then needs most of the time to balance it such that it becomes optimal or near optimal. We ended the training after round about $600,000$ episodes when there was no more improvement measureable. To evaluate the final strategies of both the HSMQ and Flat RL architectures we tested the final strategies over $10,000$ episodes without any learning enabled and compared the results to the handcoded strategies, see Table 4. It is clearly obvious that the RL strategies outperform the analytic Evade agent in all categories. While Evade is unable to prevent too many collisions, it is even slower in the overall performance due to the great safety distance that it aspires. Especially the HSMQ architecture is able to prevent almost all collisions: In the presented testruns only a single episode failed. This confirms the success of the simplification of the state space that is achieved through a hierarchical organization of the task.

5.3 Unknown Environments

The previous results already show that RL application for robot navigation is very promising, yet for a successful application of the strategies in unforseen environments or even commercial robots, the universal validity of the strategies has to be shown. For this purpose we took the trained agents and tested them under new circumstances. The results for these tests are shown in Table 5, again we display the average results of $10,000$ episodes.

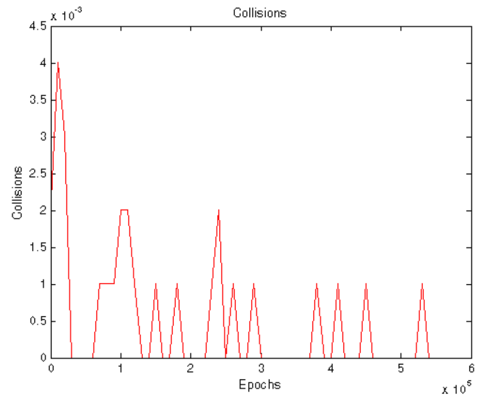


Figure 10. Graph showing the graph of the collision rate of the HSMQ configuration during the training. The first datapoint was measured after the first $10,000$ episodes have been trained.

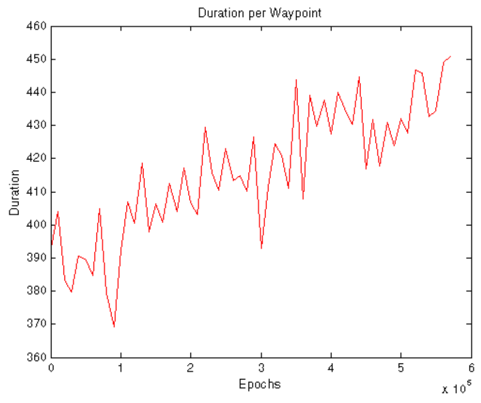


Figure 11. Graph showing the graph of the duratoin per waypoint of the HSMQ configuration during the training. The first datapoint was measured after the first $10,000$ episodes have been trained.

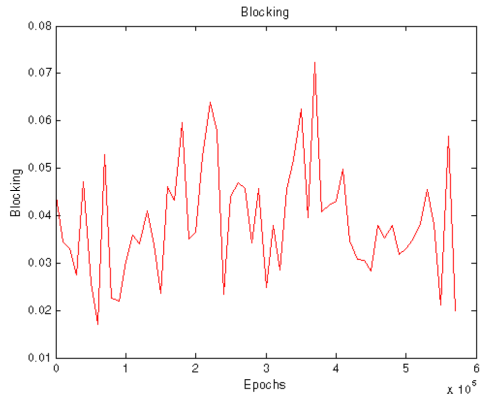


Figure 12. Graph showing the graph of the blocking rate of the HSMQ configuration during the training. The first datapoint was measured after the first $10,000$ episodes have been trained.

Map	Agent	Collisions (%)	Duration	Blocking (%)
Arena, 2 Inhabitants	Flat RL	0.04	264.88	0.02
	HSMQ	0.07	171.67	0.01
	Evade	0.09	495.30	2.16
	Planner	26.50	86.41	12.80
	Random	98.83	4, 964	7.71
Arena, 5 Inhabitants	Flat RL	0.85	1, 637.7	0.02
	HSMQ	1.29	457.73	0.09
	Evade	4.26	1, 524.8	0.04
	Planner	39.42	86.72	12.71
	Random	99.01	1, 213.2	6.87
Simplerooms, 1 Inhabitant	Flat RL	0^*	315.77	0.55
	HSMQ	0^*	297.64	0.99
	Evade	0.09	322.98	4.90
	Planner	32.53	194.99	4.12
	Random	98.93	3, 725	5.96
Section, 4 Inhabitants	Flat RL	0.91	778.13	14.66
	HSMQ	0.25	670.31	6.36
	Evade	2.58	944.35	12.81
	Planner	42.65	438.76	2.28
	Random	84.69	64, 166	0.57

Table 5. Results of the previously trained agents under new circumstances compared to the handcoded solutions. Values marked with $*$ have been verified in $50,000$ episodes.

The results indicate that the RL strategies developed a strong generalization for various cases. In all tests both RL agents outperform the handcoded Evade agent and most times the categories of second order are beaten as well. On the map Section the agents have problems with blocking the inhabitants, as the corners in the floors present unseen structural properties: The agent drives to these corners to evade an inhabitant but the inhabitant follows and waits for the agent to move out of the way. Allthough in our tests this would be counted as a blocking situation and training the agents on a map that also owns this properties is expected to prevent this, in a real environment such behavior can be acceptable, as a human is now able to pass the robot without problems.

5.4 Failure Analysis

Allthough the agents perform already very good in these tests there are still rare situations in which the agent causes a collision that might pose a danger in a real world application. It is unclear whether other state encodings or further value function approximation optimization would be able to produce a strategy that is without any flaws in this sense. Instead it is to be expected that in a real world application the agent would face more situations that are not solvable. E.g. we would expect the agent to behave correctly when humans surround the agent on purpose to test his reaction or similiar cases. The flawless performance of the agents when only one inhabitant is present (Simplerooms with one inhabitant, see Table 5) and a manual analysis of the various failure sitations (see e.g. Fig. 13) imply this as well. For this reason we recommend some kind of emergency system that stops the agent when a collision is imminent or some safe distance to the humans is crossed. With a system of this kind the effectiveness of RL navigation can be used while keeping the inhabitants safe at all time.

6 Summary

In this paper we presented our approach and the most competetive setups for training promising Reinforcement Learning strategies for safe robot navigation in home environments with respect to the inhabitants. We show that state exploration is an effective way of extracting the essence of a given situation that the agents can use

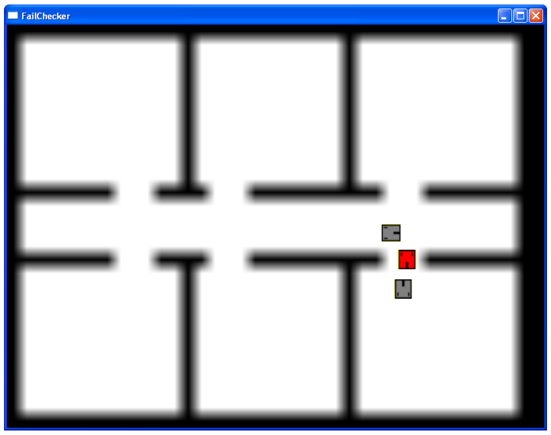


Figure 13. Example for a typical collision of the RL agents: Evading was not possible.

to develop effective yet safe strategies that outperforms the created handcoded solutions and that are independent of the map they were trained on or the number of inhabitants around. Rare collision situations can be prevented by adding an emergency stop system that activates when a security distance is breached or a similiar condition is met. For further information we refer to [4] where we thoroughly explore and compare several popular RL algorithms, different state encodings, the various parameters, function approximation options and much more.

REFERENCES

[1] A.D. Anastasiadis, G.D. Magoulas, and M.N. Vrahatis, ‘New globally convergent training scheme based on the resilient propagation algorithm’, *Neurocomputing*, **64**, 253–270, (2005).

[2] G. Biggs, T. Collett, B. Gerkey, A. Howard, N. Koenig, J. Polo, R. Rusu, and R. Vaughan. The player project, 2010. [Online; Stand 6. Januar 2010].

[3] T.G. Dietterich, ‘An overview of MAXQ hierarchical reinforcement learning’, *Lecture notes in computer science*, **1864**, 26–44, (2000).

[4] Ruben Gerlach, *Roboternavigation unter Berücksichtigung menschlicher Bewegungsabläufe*, Master’s thesis, Technische Universität Berlin, 2010.

[5] M. McPartland and M. Gallagher, ‘Creating a multi-purpose first person shooter bot with reinforcement learning’, in *IEEE Symposium on Computational Intellegence and Games*, (2008).

[6] J.E. Poliscuk, ‘The Machine Learning Method: Analysis of Experimental Results’, *Journal of Quantitative Linguistics*, **11**(3), 215–232, (2004).

[7] M. Ponsen, P. Spronck, and K. Tuyls, ‘Hierarchical reinforcement learning with deictic representation in a computer game’, in *Proceedings of the BNAIC*, (2006).

[8] M. Riedmiller and H. Braun, ‘RPROP-A fast adaptive learning algorithm’, *Proc. of ISCIS VII*, (1992).

[9] M. Riedmiller, T. Gabel, F. Trost, and T. Schwegmann, ‘Brainstormers 2D-Team Description 2008’, (2008).

[10] G.A. Rummery, *Problem solving with reinforcement learning*, Citeseer, 1995.

[11] A.A. Sherstov and P. Stone, ‘Function approximation via tile coding: Automating parameter choice’, *Lecture notes in computer science*, **3607**, 194, (2005).

[12] S.P. Singh and R.S. Sutton, ‘Reinforcement learning with replacing eligibility traces’, *Machine Learning*, **22**(1-3), 123–158, (1996).

[13] P. Stone, R.S. Sutton, and G. Kuhlmann, ‘Reinforcement learning for robocup soccer keepaway’, *Adaptive Behavior*, **13**(3), 165, (2005).

[14] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1. aufl. edn., 1998.

About the evolution of neural control for walking machines

Jörn Fischer and Thomas Ihme[1]

Abstract. Walking machines are complex, fragile systems, where the complexity of its behaviour seems to determine the complexity of its controllers. When having a closer look at simple animals the number of neurons responsible for locomotion is still large. Different works try to extract the essence of such ”real life” controllers in order to implement them on robot platforms [12, 3]. If the morphology of the robots body is quite different from the one of animals there is another promising approach: the evolution of a neuro controller in a simulated world with a simulated robot [10, 7, 1, 2]. In the following article we present a method to evolve and to reuse neuro-controllers keeping the complexity of the networks and the expense to evolve them small. We analyze the neural entities in order to understand them and to be able to prevent an unwanted behavior. As a last proof the resulting networks are tested on the real walking machine to find out the limitations of the physical simulation environment as well as of the neural controller itself. Our real platform is a four-legged walking machine called Hodgkin. The task it performs is an obstacle avoidance behavior in an unknown environment.

1 Introduction

Evolving and understanding efficient neural controllers for physical walking machines is still a challenge in the area of robotics [10]. Most researches do not concentrate on sensor driven neural control of legged machines. The walking robots are solely designed to perform various motions without sensing the environment. Two articles [6],[4] report on an evolutionary approach to create neural controllers for locomotion and obstacle avoidance in an unknown environment. Both articles lead to a similar result: The motor neurons of the legs on the oposite site of a detected obstacle are inhibited. The walking robot then tends to walk a slight curve to avoid the obstacle. The robot fails in narrow corners or when detecting a single cylindric obstacle in front of itself.

Our aim is to evolve small, simple and robust networks, which can be coupled together in order to get a more complex behavior. This modular neurodynamics approach is used together with a physical simulation environment to evolve controllers for real world tasks such as obstacle avoidance. In this article this procedure is demonstrated for a four-legged robot, called Hogkin. The following chapter describes the equations used to simulate the neural network. Then the main technical specifications of the platform are given. The next chapter starts with the evolution of behavior such as locomotion and obstacle avoidance in a physical simulation environment, which then leads to an implementation on the real platform. Finally, the results are discussed and an outlook is given.

[1] Hochschule Mannheim, Institut fur Robotik, Paul-Wittsack Str. 10, 68163 Mannheim, Germany, email: {j.fischer|t.ihme}@hs-mannheim.de

2 A simple time discrete neuron

Biological neurons cover an enormous complexity. To simulate such neurons in detail would assume a lot of specifications about the types of neurons and their behavior, while even simple neural interaction would hardly be analyzable.

The neural model we work with is much simpler and enables us to understand even small entities with recurrent connections. The equation for the activity of such a discrete-time neuron is written down as follows:

$$\varphi_j^{t+1} = \theta_j + \sum_{i=1}^{n} w_{ji}^t o_i^t(\varphi_j^t) \quad (1)$$

Here φ_j denotes the activity of neuron j, $w_{ji} \in \mathbb{R}$ is the weight from neuron i to neuron j, θ_j is the bias term and $o_i^t \in \mathbb{R}$ is the output activity of the neuron i at time $t \in \mathbb{N}$. The output as a function of φ_j^t is given by the transfer function, which we choose the hyperbolic tangent function:

$$o_j^t(\varphi_j^t) = \sigma(\varphi_j^t) = tanh(\varphi_j^t) \quad (2)$$

This neural model is simple but sufficient to be able to generate even complex behavior. The only limiting factor is the number of neurons and interconnections simulated.

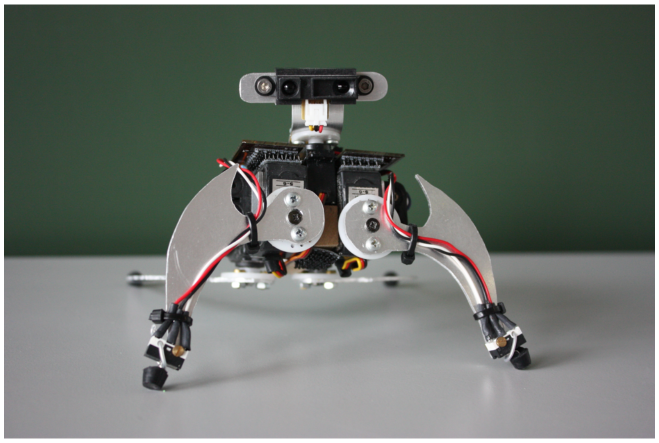


Figure 1. The walking robot Hodgkin with four legs, five motors and an ”artificial brain” of 19 neurons.

3 The walking machine Hodgkin

Hodgkin is an unusual four-legged robot, where each leg has only one degree of freedom. Each joint is controlled by a strong full metal

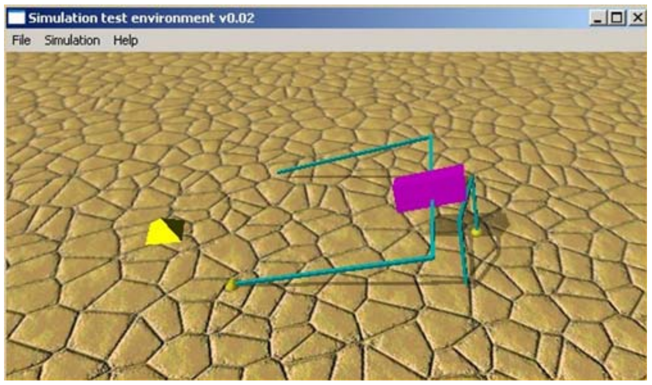


Figure 2. The simulated robot in a first step trying to walk straight.

servomotor. The levers are kept short and the body is narrow to ensure optimal force exploitation. The building blocks are kept modular to be able to change components very quickly and to enable the construction of different morphologies. Each of Hodgkins front legs has a foot contact sensor. In addition an infrared sensor is placed on a pivoting head on the front side of the robots body. All in all Hodgkin has 5 active degrees of freedom and 3 sensors. The control of the robot is kept on a simple but powerful board, the "mini MBoard", which is able to control up to 16 motors, and which has 8 analog sensor inputs and a size of 40 mm x 34 mm.

4 Evolving neural entities for walking

To use an evolutionary approach effectively it is necessary to have a physics simulation environment, which simulates the robot together with the environment as fast and as exact as possible. Our simulator is based on ODE (Open Dynamics Engine)[2] and enables an implementation, which is faster than real time and which is precise enough to represent the corresponding behavior of a real robot. This simulation environment is connected to our Evolution Program (EP). In this program it is possible to preserve parts of a neural network which already has an appropriate performance, it is possible to fix these structures, and to continue the evolution by adding neurons or connections or removing parts of the network which are not yet fixed.

Once the model of Hodgkin is implemented in the physics simulator, a suitable fitness function has to be defined. In the simplest case it may result from the distance the robot walks during a given time interval. The desired neurocontroller for the pure walking task has no input neurons and 4 output neurons providing the motor signals. For simplicity the connections between output neurons are suppressed in the first experiment. The simulation environment is a simple plane on which Hodgkin should learn to walk along a straight line. The simplest solution for the hidden layer was found to be a quasi-periodic oscillator composed of two neurons similar to the one discribed in [11]. By using symmetric output weights a typical walking gate was obtained, which enabled an efficient forward movement.

5 Adding the "MRC" controller for Obstacle Avoidance

For the second task, obstacle avoidance, is endowed with an additional motor, turning an infrared sensor left and right in an oscillatory motion. Furthermore, one has to insert obstacles into the simulated environment. The difficulty with environments for obstacle avoidance is to define a suitable fitness function. The robot should move as far as it can, but to avoid obstacles may start to walk in circles. This should of course be prevented. The simplest way to define the fitness is to take the Euclidean distance from the start to the end point of the robots trajectory, and to let it run in an environment with walls. We choose an environment like a room with 4 walls, where one wall is randomly left out for each run. In this environment the robot has to avoid the walls and to turn inside of the virtual room until it finds the exit to escape. To maximize the fitness the robot should then continue walking a straight line to get as far as possible from the starting point. Evolving a neuro-controller leads to the following solution: The robot prevents hitting the walls by turnig in only one direction, while observing the obstacle. This is done by inhibitting the motor neuron of one of the forward pushing legs when an obstacle is detected. Further experiments show that the robot sometimes gets stuck in narrow corners. The simple neuro controller is not able to turn without forward movement, and it is not able to wak backwards.

In earlier works of M.Hülse and F.Pasemann [8] a neuro-controller called "MRC" is evolved, which enables a wheeled Khepera robot to avoid obstacles in an elegant way. So the idea is to fuse the neuromodule "MRC" with our resulting controller. To enable even backwards walking we have to construct a network which in principle performs a kind of multiplication between two signals, the quasiperiodic oscillator and the infrared distance-sensor signal (as in [9]). The output then is directly connected with the forward pushing back legs. Tuning the resulting networks weights with the Evolutionary Program results in a quite well performing neuro controller (Fig. 3) which enables the simulated robot to avoiding obstacles in an elegant and efficient way. Testing this controller on the hardware platform Hodgkin leads to an astonishing similar behavior, where differences between simulation and real hadware are hardly to be seen.

In a last experiment we compare the performance of the neural network to the one of a simple 2 layered subsumption architecture. The lower level imlements a forward movement, while the upper level inhibits the lower level when detecting an obstacle and turns the robot away from it as long as it "sees" the obstacle. Both controllers should make the robot walk through a typical robocup rescue scenario with cylindric obstacles. The neural controller just reaches from the start to the end, while the subsumption controller often gets stuck in front of the cylindric obstacles, turning the robot to the left, then to the right, then to the left again etc.

6 Conclusions

With a physics simulation environment and our Evolution Program we evolve neuro modules for locomotion and for obstacle avoidance of a new walking platform. The process should be started with simple forward movements. The resulting network will be analyzed and decided which weights should be fixed. After increasing the task difficulty the evolution is continued by changing the neural connection weights and by evaluating the new individuals. We obtain small and robust networks, which may be understood in their functionality as well as in their dynamical properties. Understanding such neuro modules, we are able to couple these modules together to improve the robots performance. This method is demonstrated on the four-legged robot Hodgkin successfully performing obstacle avoidance in various environments. Future research will concentrate on evolving neuro-controllers with modulatory learning capabilities [5] to enable the robot to adapt to its environment.

[2] see also: http://opende.sourceforge.net/

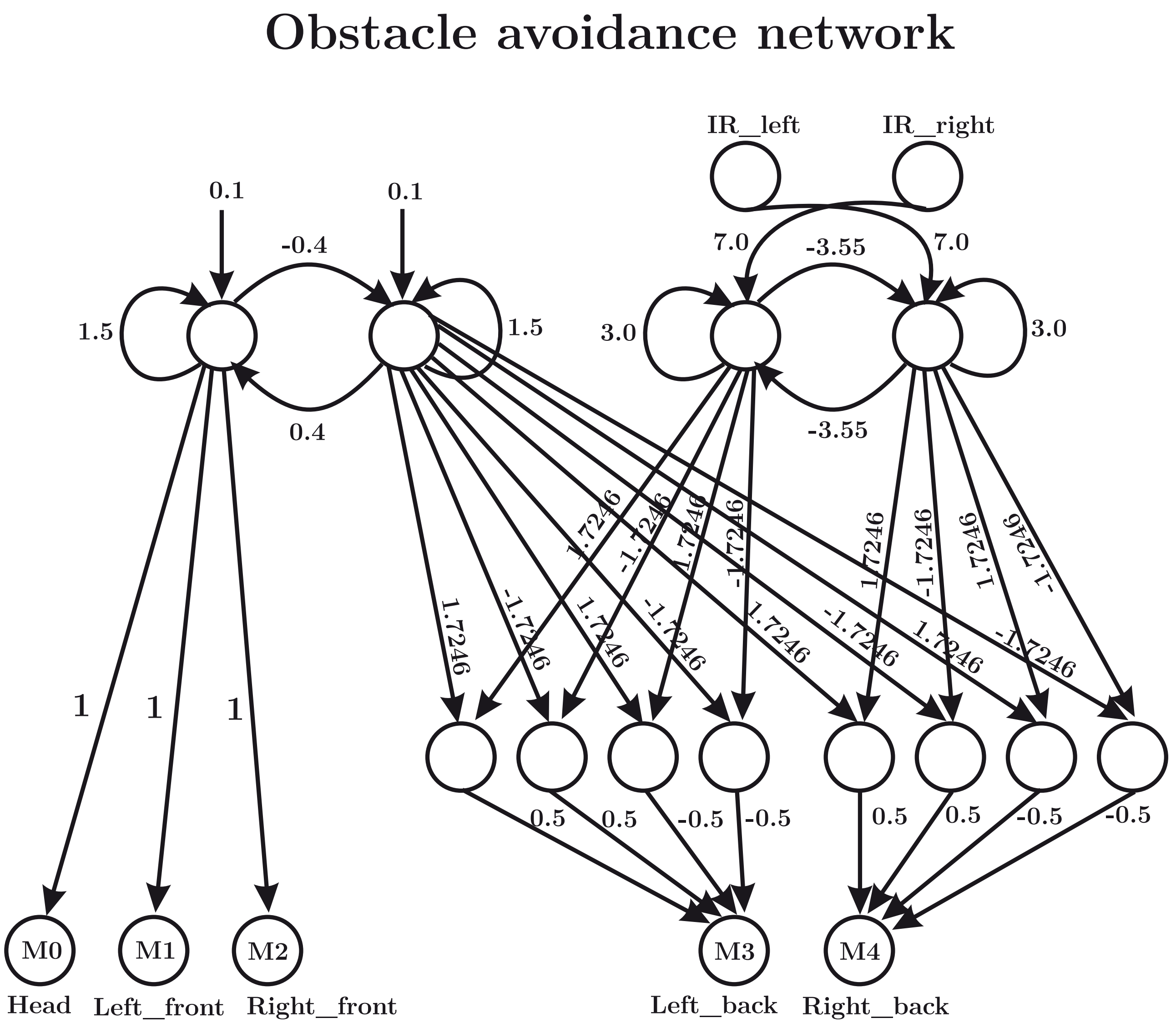


Figure 3. Hodgkins "artificial brain" consisting of 19 neurons. The upper left shows a neural oscillator needed for simple walking, the upper right, the "MRC"-module evolved by Frank Pasemann and Martin Hülse, is needed for obstacle avoidance and the lower right corner shows the multiplying network which enables walking in different directions.

7 Acknowledgements

Special thanks to Ralph Breithaupt, Poramate Manoonong and Frank Pasemann, who helped with fruitful discussions.

REFERENCES

[1] R.D. Beer, H.J. Chiel, and J.C. Gallagher, 'Evolution and analysis of model cpgs for walking ii. general principles and individual variability', *J. Computational Neuroscience*, **7**(2), 119–147, (1999).

[2] A. Calvitti and R.D. Beer, 'Analysis of a distributed model of leg coordination. i. individual coordination mechanisms', *Biological Cybernetics*, **82**, 197–206, (2000).

[3] H. Cruse, J. Dean, T. Kindermann, J. Schmitz, and M. Schumm, 'Walknet - a decentralized architecture for the control of walking behavior based on insect studies', in *Hybrid Information Processing in Adaptive Autonomous Vehicles*, ed., G. Palm, Springer, (1999).

[4] D. Filliat, 'Incremental evolution of neural controllers for navigation in a 6legged robot', in *Fourth International Symposium on Arti?cial Life and Robotics*, (1999).

[5] J. Fischer, *A Modulatory Learning Rule For Neural Learning And Metalearning in Real World Robots with Many Degees of Freedom*, Ph.D. dissertation, Westfälische Wilhelms Universitt Münster, 2003.

[6] J. Fischer, F. Pasemann, and P. Manoonpong, 'Neuro-controllers for walking machines - an evolutionary approach to robust behavior', in *7th Int. Conf. on Climbing and Walking Robots*, (2004).

[7] Frédéric Gruau, 'Automatic definition of modular neural networks', *Adapt. Behav.*, **3**(2), 151–183, (1994).

[8] M. Hülse, 'Dynamical neural schmitt trigger for robot control', in *ICANN 2002*, ed., J.R. Dorronsoro, LNCS2415, pp. 783–788, (2002).

[9] P. Manoonpong, F. Pasemann, and J. Fischer, 'Modular neural control for a reactive behavior of walking machines', in *Proceedings of the CIRA2005, Helsinki University of Technology, Finland*, volume ISBN: 0-7803-9355-4, (2005).

[10] G.B. Parker and Z. Li, 'Evolving neural networks for hexapod leg controllers', in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, pp. 1376–1381, (2003).

[11] F. Pasemann, M. Hild, and K. Zahedi, 'So(2)-networks as neural oscillators', in *Biological and Artificial computation: Methodologies, Neural Modeling and Bioengineereng Applications, IWANN 7'th*, Lecture Notes In Computer Science, (2003).

[12] J. Schmitz, J. Dean, Th. Kindermann, M. Schumm, and H. Cruse, 'A biologically inspired controller for hexapod walking: simple solutions by exploiting physical properties', *Biological Bulletin*, **200**(2), 195–200, (April 2001).

Offline and active gradient-based learning strategies in a pushing scenario

Sergio Roa and Geert-Jan Kruijff
German Research Center for Artificial Intelligence / DFKI GmbH
{sergio.roa,gj}@dfki.de[1]

Abstract.
When operating in the real world, a robot needs to accurately predict the consequences of its own actions. This is important to guide its own behavior, and in adapting it based on feedback from the environment. The paper focuses on a specific problem in this context, namely predicting affordances of simple geometrical objects called polyflaps. A machine learning approach is presented for acquiring models of object movement, resulting from a robot performing pushing actions on a polyflap. Long Short-Term Memory machines (LSTMs) are used to deal with the inherent spatiotemporal nature of this problem. An LSTM is a gradient-based model of a Recurrent Neural Network, and can successively predict a sequence of feature vectors. The paper discusses offline experiments to test the ability of LSTMs to solve the prediction problem considered here. Cross-validation methods are applied as a measure of convergence performance. An active learning method based on Intelligent Adaptive Curiosity is also applied for improving the learning performance of learners trained offline, generating a combination of learners specialized in different sensorimotor spaces after the knowledge transfer.

1 Introduction

Robots need to learn in continuously changing environments. One way to learn from the world is by interacting with objects present in it. This work is inspired by the fact that humans and animals in general are able to properly adapt to a dynamic environment. Theories of cognitive development like the theory of affordances [5] attempt to explain how creatures are able to acquire sensorimotor skills when they are faced with the different features found in the environment. For instance, surfaces afford posture, locomotion, collision, manipulation, and in general behaviour [5]. Particular objects can afford sliding, flipping, rolling behaviour. A consequence of this is that the creature should then properly predict the consequences of actions on a surface given its own body configuration. We consider here a special case of affordances learning in robots, namely that of predicting consequences of pushing simple geometrical objects called *polyflaps*. Polyflaps have been proposed to design simple learning scenarios. A polyflap is a polygon (concave or convex) cut out of a flat sheet of some material (e.g. cardboard) and folded once (anywhere) to produce a 3-D object [17], cf. Fig. 1. By combining different objects and performing different actions, we can steadily increase the complexity of the learning environment.

[1] The research reported of in this paper is supported by EU FP7 IP "CogX" (ICT-215181)

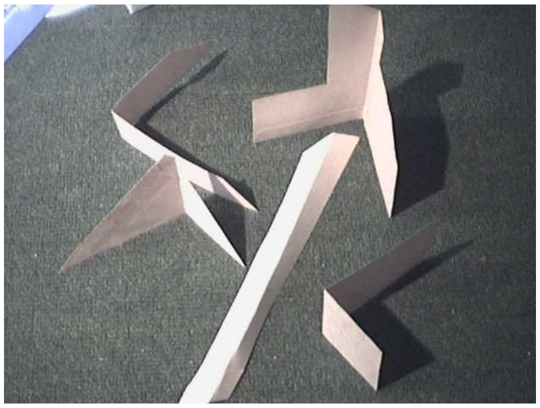


Figure 1. Polyflaps, http://www.cs.bham.ac.uk/∼axs/polyflaps/. Used here are polyflaps of the shape (bottom-right corner)

In this paper we discuss a learning scenario where a simulated robotic arm interacts with a polyflap. In the implementation we use the NVidia® PhysX™ library that allows us to perform realistic physical simulations and to obtain 3-dimensional feature vectors, so that we can easily re-adapt our algorithms to real scenarios. Although providing an idealized scenario, these experiments are necessary to establish a base line from which we can start facing noisy and incomplete features, where learning machines should be able to generalize and present outcomes in the presence of uncertainty. The learning machines we use are able to process spatio-temporal features. Specifically, we use the Long Short-Term Memory (LSTM) [7, 6] model of an Artificial Neural Network. The main objective is that the robot arm pushes the object and predicts a sequence of polyflap poses encoded as rigid body transformations during a certain time interval following the pushing action. To reduce the space- and time complexity of the problem, we select a discrete set of possible actions and starting positions for the arm to start the pushing movement. This reduction of dimensionality affords us also to evaluate and analyse more easily and carefully the learning algorithms and its corresponding results. In general, *sliding* and *flipping* affordances are obtained by applying pushing actions. The experiments show that the machines are able to model a sort of regression function that fits the data very accurate. This fact is also crucial from the point of view of dimensionality reduction, since the use of a learning machine together with its generalization abilities can highly reduce the need of storage space. Moreover, the inherent recurrent topology of these networks affords the reduction of space needed for storing spatio-temporal information.

The characteristics of these learning machines are appropriate for autonomous development of robots [10, 14]. Robots should be able to autonomously acquire sensorimotor skills by interaction with the environment. Thus, machines that are able to learn in an online and active manner need to be used. Neural Networks in general are useful for these tasks, since their weights can be updated efficiently by using one forward and one backward pass when we use gradient-based methods. However, one has to be careful with the problem of overfitting data (bias-variance tradeoff). Therefore, a sufficiently big set of samples and iterations are needed in order to generalize sufficiently well a dataset.

We tested the topology of the neural network in order to find a good compromise between computational complexity and generalization ability. For that purpose, we extracted n-fold crossvalidation sets and analyzed the average sum of squares error for all training epochs. The problem that we tackle can be regarded as a time series prediction problem approached by regression techniques. Therefore, the sum of squares error is a good performance estimation. The experiments show that the machines are able to accurately predict a feature vector, given a history of precedent feature vectors that together form a sequence. After offline training, we applied an active learning technique based on the Intelligent Adaptive Curiosity algorithm [10, 14], by including an additional set of actions in order to test the autonomous generation of different regions in the sensorimotor space that allows an active selection of samples via maximization of a measure of learning progress and multiple learners specialized in each region. We also show that the generalization is improved by the set of machines (which are only “biased” for their corresponding regions).

This paper is organized as follows. In the next section, we present a current state of the art related to affordances learning in robots and recurrent neural networks. In section 3, we describe the learning scenario and the features we used for training LSTMs. In section 4 we present the offline learning mechanism and architecture employed. In section 5 we show and explain experimental results for offline experiments with LSTMs. In section 6 we explain the active learning mechanism and results and in section 8 we present some concluding remarks and planned work.

2 Related Work

Affordances learning has been introduced in the field of robotics in recent years. The reason is that aiming to autonomous behaviour in robots requires an inspiration from biological cognitive systems, which are very successful on acquiring sensorimotor skills by their own means. As a cognitive science theory, the field was introduced by the perceptual psychologist J.J. Gibson [5]. An affordance in this sense is a resource or support that the environment offers an agent for action, and that the agent can directly perceive and employ. From the robotic perspective, this concept implies that the robots should be able to predict consequences of actions given certain object features and robotic embodiment.

In the field of robotics, a compilation of works related to affordance-based robot control can be found in [15]. A similar approach to the one presented in this work is described in [12]. In that work, labels of object/action pairs and 11 features encoding the action performed and the object behaviour are used to train Self Organizing Maps. In this way, they cluster this space and map the features to the target function represented by such labels. Pushing actions were performed on different objects in a real environment. Other approaches have used also similar features and learning methods and have studied different kinds of affordances [2].

Perception of affordances has also been addressed with reinforcement learning techniques. In [11], the robot performs different learning stages starting from recognizing affordances and finally accomplishing some task given the affordances that the robot has already acquired in earlier stages. In that work, liftable vs. non-liftable objects are recognized and Markov Decision Processes are used for the goal-based task. In [10, 14] the robot autonomously enters different stages of development by interacting with objects or performing some action, which is selected according to a measure of “interestingness”. Thus, robots are intrinsically motivated to perform actions that offer an opportunity to learn according to an estimation of learning progress calculated from prediction error histories.

However, we can consider that these aproaches use a kind of short-term memory or mapping approach that does not take into account the spatio-temporal processing of data when an action is performed in a given time interval. Moreover, in some approaches there is an explicit labelling of the recognized affordances or the robot has no means to evaluate the accuracy of its predictions. In order to evaluate the abilities of learning machines in processing a series of features like rigid body transformations that gives us a more accurate assessment of the object poses and behaviour, we are using recurrent neural networks that are known to process sequences and obtain proper generalizations by infering regression functions.

A simulated scenario using also polyflaps is described in [9]. The authors formalise the learning problem in a probabilistic framework. Explicit 3D rigid body transformations are predicted by that models and they are tested against novel objects similar in shape to polyflaps.

Long Short-Term Memory machines have been used for problems like time-series prediction, sequences classification, phoneme classification, reinforcement learning, among others [7, 6, 1]. They are appropriate to handle long-term dependencies in data sequences. Therefore, they seem to have a high potential to be used in learning tasks where compositionality and conditional dependencies of events or states is encountered through a relatively broad time period.

The work described in this paper is a follow-up of the one presented in [13].

3 Learning Scenario

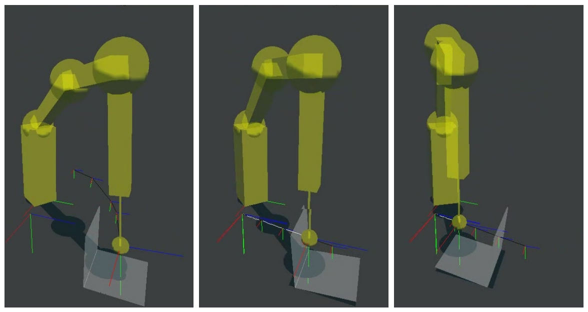


Figure 2. Learning scenario with a polyflap

The learning scenario is shown in Fig. 2. The simulated arm corresponds to a Neuronics® Katana 6M^{TM} arm with a ball as a simple finger. In order to simulate a pushing action we apply a linear trajectory over a specified time period until it reaches the desired pose. The arm has 6 joints, including the last joint for the finger which is static. The representation of object poses are in Euler angles with respect to a reference frame which is the origin in the scene (6-D pose).

The features corresponding to the arm are a starting 6-D pose vector for the end-effector $\mathbf{e}_0$, and a real value denoting a direction angle Θ ranging from 60 to 120 degrees, parallel to the ground plane in the direction to the center of the standing polyflap side. Together, these features form the motor command feature vector denoted as $\mathbf{m}$. The values are all normalized to obtain vectors with mean 0 and standard deviation 1.0. A 6-D pose vector corresponding to the polyflap pose is denoted as $\mathbf{p}_t$ at time t. The pose $\mathbf{p}_0$ is fixed for all experiments.

Then, the concatenation $\mathbf{f}_0 = [\mathbf{m}\ \mathbf{e}_0\ \mathbf{p}_0]$ represents the feature vector to be fed initially to the neural network. The subsequent feature vectors fed to the machine have the form $\mathbf{f}_t = [\mathbf{0}\ \mathbf{e}_t\ \mathbf{p}_t]$, where the size of $\mathbf{0}$ is the size of $\mathbf{m}$. This representation affords the learning machine to attain a better convergence.

During the execution of the arm path, we obtain a series of poses $\langle \mathbf{p}_t, \mathbf{e}_t \rangle$ to construct a feature vector $\mathbf{f}_t$. We extract then n polyflap and effector poses and finally we build a sequence set $S = \{\mathbf{f}_{t=1}^{n}\}$. So, a particular sequence set (an instance) is used in each iteration of the experiment to be fed to the LSTM in $n+1$ steps. For the time step t, a training tuple $\langle \mathbf{f}_t, \mathbf{t}_t \rangle$ is used for the neural network learning procedure, where the feature vector $\mathbf{f}_t$ represents the input vector and $\mathbf{t}_t = \mathbf{p}_{t+1}$ the target (predicted) vector encoding the predicted polyflap pose.

This representation then encodes the rigid body transformations of polyflap and effector through these n steps and also encodes the given robot control command that performs the pushing movement. In order to discretize and reduce the dimensionality of the task, we only used a discrete number of different starting positions for the arm to start the pushing movement.

4 Offline Learning method

The learning process used for training LSTMs with the features described in section 3 is described here. As mentioned in the previous section, a dataset $\mathcal{D}$ containing a certain quantity of sequences S_i is obtained and we perform offline experiments with these data.

A LSTM machine is usually composed of an input layer, a hidden layer and an output layer. In general, recurrent neural networks can have recurrent connections for all their neurons. In particular, in this work we only use recurrent connections for the hidden layers. We also made preliminary experiments with networks with no recurrent connections and we found less performance. The LSTM [7, 6, 1] architecture was developed in order to solve some learning issues in recurrent neural networks related to long-term dependencies learning. These problems sum up to the problem that errors propagated back in time tend to either vanish or blow up. This is known as the problem of vanishing gradients.

LSTM's solution to this problem is to enforce *constant* error flow in a number of specialized units, called Constant Error Carrousels (CECs), corresponding to those CECs having linear activation functions not decaying over time. CECs avoid to transmit useless information from the time-series by adding other input gates that regulate the access to the units. Thus, they learn to open and close access to the CECs at appropriate moments. Likewise, the access from the CECs to output units is controlled by multiplicative output gates and they learn in a similar way how to open or close the access to the output side. Additionally, forget gates [3] learn to reset the activation of the CECs when the information stored in them is no longer useful, i.e., when previous inputs need to be forgotten. The combination of a CEC with its associated input, output and forget gate is called a memory cell, as depicted in Fig. 3. Other additions are peephole weights [4], which improve the LSTM's ability to learn tasks that require precise timing and counting of internal states, and bidirectional connections [16].

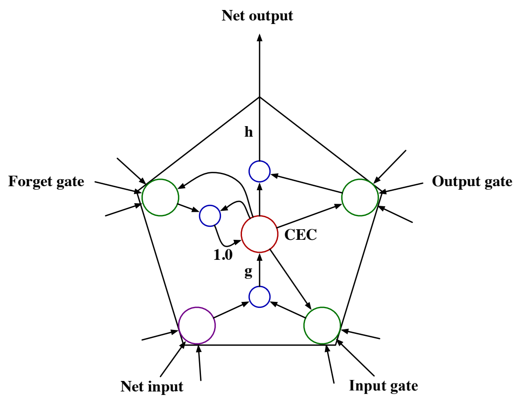


Figure 3. LSTM memory block with one cell. The internal state of the cell is maintained with a recurrent connection of fixed weight 1.0. The three gates collect activations from inside and outside the block, and control the cell via multiplicative units (small circles). The input and output gates scale the input and output of the cell while the forget gate scales the internal state. The cell input and output activation functions (g and h) are applied at the indicated places [6].

In this work, we used 10 memory blocks in the hidden layer, which was found to be a good compromise between computational complexity and convergence.

When some input vector is fed to the network, the forward pass is calculated as follows. Let us denote an output neuron (unit) activation y^o, an input gate activation y^{in}, and output gate activation y^{out} and a forget gate activation y^f. Then, for the time step t each of them are calculated in the following standard way:

$$y^i(t) = f_i(\sum_j w_{ij} y^j(t-1)), \tag{1}$$

where w_{ij} is the weight of the connection from unit j to unit i, and f the activation function. In this paper, we only consider one CEC activation (one cell) for each memory block. The CEC activation s_c for the memory cell c is computed as follows:

$$s_c(t) = y^{f_c}(t) s_c(t-1) + y^{\text{in}_c}(t) g(\sum_j w_{cj} y^j(t-1)), \tag{2}$$

where g is the cell input activation function. The memory cell output is then calculated by

$$y^{s_c}(t) = y^{\text{out}_c}(t) h(s_c(t)), \tag{3}$$

where h is the cell output activation function. The backward pass is a steepest (gradient) descent method which updates the weights of the different types of units. Consider a network input $a_j(t)$ to some unit j at time t. In general, the gradient is defined as:

$$\delta_j(t) = \frac{\delta E}{\delta a_j(t)}, \tag{4}$$

where E is the objective (error) function to be minimized and used for training. For a detailed explanation of the backward pass equations for each unit type cf. [6]. Since we are dealing with a regression problem, we consider the sum of squares error as a performance measure. The error function is defined as:

$$E_t = \frac{1}{2K} \sum_i (y_i - y'_i)^2, \tag{5}$$

where K is a normalization factor which depends on the size of each sequence n_i and the total number of sequences in the dataset k. y_i is the output unit activation and y'_i is the expected value. The learning process is described in the Algorithm 1.

```
Data: A dataset $\mathcal{D}_1$ containing $k$ sequences of variable size $n_i$
      for training. A dataset $\mathcal{D}_2$ containing $z$ sequences of size
      $n_j$ for testing.
Result: An LSTM machine after error minimization.
Nr. of epochs $ep = 0$.
repeat
    for $i$=1 to $k$ do
        for $j$=1 to $n_i$ do
            Input: Present training tuple $\langle \mathbf{f}_{ij}, \mathbf{t}_{ij} \rangle$ ($j$th forward
                   pass step).
        end
        Calculate error $e_i$ associated to current training
        sequence $S_i$.
        Backward pass.
    end
    Evaluate error $E_t$ with the test set $\mathcal{D}_2$.
    Epoch $ep = ep + 1$.
until No new network found with lowest error after 20 epochs ;
```
Algorithm 1: Offline learning process

For the purpose of calculating the number of training sequences that are necessary so that convergence improves, we generated n-fold cross-validation sets. We split a dataset $\mathcal{D}$ into n disjoint sets of equal size that are used for testing. We used the remaining data for training n different networks.

5 Experimental results for Offline Learning

In order to test the convergence of LSTMs we used 10-fold cross-validation sets for three different dataset sizes, namely 100, 200 and 500. That allowed us to estimate the approximate number of samples that are needed to learn with high precision the prediction task.

In Fig. 4 a comparison of the average sum of squares error (SSE) and SSE standard deviation is shown. In this case, the SSE is averaged among all the cross-validation sets. The picture shows that the SSE is considerably reduced when more samples are used, as expected, and likewise the standard deviation of the SSEs.

6 Active Learning

The active learning procedure is based on the work of Oudeyer et al. [10] about Intrinsic Motivation Systems. The general idea of the Intelligent Adaptive Curiosity (IAC) algorithm is that a meta-learning system samples a set of actions and selects one that maximizes the learning progress, which is a measure based on the difference between smoothed current and previous mean error quantities. The learning progress $\mathcal{L}_r$ is associated to a region $\mathcal{R}_r$ in the sensorimotor space. Starting with one region, successive regions are obtained by splitting the sensorimotor space depending on a measure

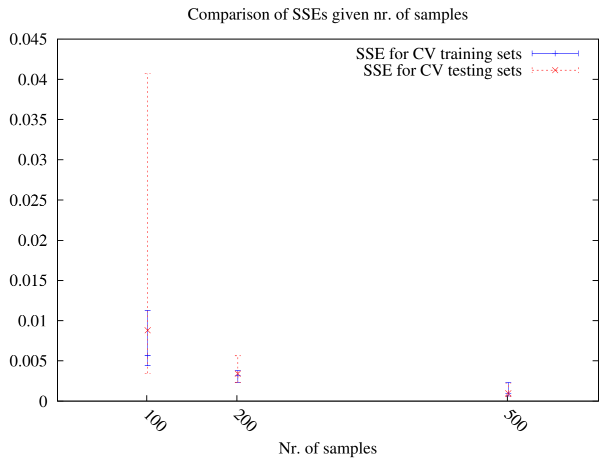


Figure 4. SSEs are reduced when increasing the dataset size.

of variance in the dataset $\mathcal{D}_r$ (exemplars used for Region $\mathcal{R}_r$). This division is performed after $|\mathcal{D}_r|$ achieves a certain threshold κ. A dataset $\mathcal{D}_r$ for a Region R_r is split in two datasets $\mathcal{D}_{r+1}, \mathcal{D}_{r+2}$ (for regions $\mathcal{R}_{r+1}, \mathcal{R}_{r+2}$). Let us denote

$$\mathcal{D}_r = \{S_i\}$$

the set of instances in region $\mathcal{R}_r$. Then the split of $\mathcal{D}_r$ defined by the index c with value v_c is performed when the following criterion (Γ) is met:

- all the instances S_i of $\mathcal{D}_{r+1}$ have the cth component of their motor command vector $\mathbf{m}_i$ smaller than v_c.
- all the instances S_i of $\mathcal{D}_{r+2}$ have the cth component of their motor command vector $\mathbf{m}_i$ greater than v_c.
- the quantity $|\mathcal{D}_{r+1}| \cdot \sigma(\{[\mathbf{e}_{ij}\ \ \mathbf{p}_{ij}]_{j=1}^{n_i} \in \mathcal{D}_{r+1}\}) + |\mathcal{D}_{r+2}| \cdot \sigma(\{[\mathbf{e}_{ij}\ \mathbf{p}_{ij}]_{j=1}^{n_i} \in \mathcal{D}_{r+2}\})$ is minimal, where

$$\sigma(\mathcal{S}) = \frac{\sum_{v \in \mathcal{S}} \|v - \frac{\sum_{v \in \mathcal{S}} v}{|\mathcal{S}|}\|^2}{|\mathcal{S}|}$$

where $\mathcal{S}$ is a set of vectors.

Each region stores all cutting dimension and values that were used in its generation as well as in the generation of its parent regions. For the region $\mathcal{R}_r$ a learning machine $\mathcal{M}_r$ is stored, and this machine is inherited by the child regions. The learning process is described in the Algorithm 2.

```
Data: An initial region $\mathcal{R}_0$ which encompasses the whole
      sensorimotor space.
Result: A set of regions $\{\mathcal{R}_r\}$ with corresponding LSTM
        machines $\{\mathcal{M}_r\}$.
for $i$=1 to $I$ do
    Choose a motor command action
    $\mathbf{m}_{r,i} = \arg\max_{\mathbf{m} \in \{R_r\}} \{\mathcal{L}_{r,i}\}$ among all current regions
    $\{\mathcal{R}_r\}$ by using a near to greedy policy with probability 0.3.
    if $\kappa$ then
        Split region $\mathcal{R}_r$ into $\mathcal{R}_{r+1}$ and $\mathcal{R}_{r+2}$ according to $\Gamma$.
    end
    Calculate error $e_{r,i}$ associated to current training sequence
    $S_{r,i}$.
    Update the machine $\mathcal{M}_r$ with a forward and backward pass.
    Calculate smoothed mean error $\varepsilon_{r,i+1}$ and $\varepsilon_{r,i+1-\tau}$ with a
    window parameter $\tau$ and a smoothing parameter $\theta$.
    Calculate the decrease in the mean error rate
    $\Delta_{r,i+1} = \varepsilon_{r,i+1} - \varepsilon_{r,i+1-\tau}$.
    Calculate the learning progress $\mathcal{L}_{r,i+1} = -\Delta_{r,i+1}$.
end
```

Algorithm 2: Active learning process

7 Experimental Results for Active Learning

In order to test the active learning mechanism, the main idea is to train offline a LSTM with a subset of all possible starting positions producing a partial set of actions and thus a dataset $\mathcal{D}_0 \subset P_0$, where P_0 is the sensorimotor manifold encompassing D_0. Then, we use this machine in the active learning loop allowing additional actions, so that at the end we generate a dataset $\mathcal{D}_1 \subset P_1$, where $P_0 \subset P_1$. The hypothesis is that the algorithm will start producing more frequently actions corresponding to the sensorimotor regions associated to the new actions.

Thus, we first trained offline a LSTM with a subset of possible starting positions for the arm movement and a number of sequences equal to 500. This generates the dataset $\mathcal{D}_0$. When initializing the active learning procedure, we allowed all possible starting positions for the arm movement. Then, we initialize the region R_0 with the already trained machine $\mathcal{M}_0$ that introduces better generalization performance according to the cross-validation sets. We apply a maximum number $I = 300$ of iterations, after which a new dataset $\mathcal{D}_1$ is generated. Then, we merge the datasets into a set $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1$. We use the set $\mathcal{D}$ to test the errors of the machine trained offline and the ensemble of machines trained via active learning. The results are shown in Table 1.

Table 1. An ensemble trained via IAC against an offline trained machine.

Machine	Avg SSE
Offline	0.4251
Active	0.211991

The unique observation here is that the generalization performance is improved by using the new active learner, which is a expected result. In order to check the hypothesis presented above, we analysed the learning progress of the ensemble of machines created after splitting the sensorimotor space in different regions.

As expected, the algorithm starts to select very frequently actions that are new or "interesting". In Fig. 5, we can observe the frequency of actions generated from each set of starting positions for a window of 20 iterations. For instance, from index ∼150 to ∼250 the new set of actions are more frequent. This result also confirms the generation of different stages of development that the IAC algorithm produces [10]. We make the same observation for a specific region (Fig. 6). In Fig. 7 the curves of learning progress and error for the corresponding region are shown. We can observe that the learning progress curve rises and the error drops.

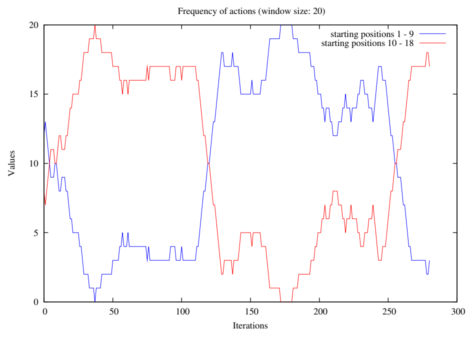


Figure 5. Frequency of actions in the experiment.

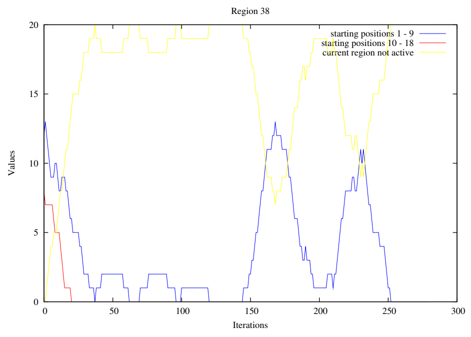


Figure 6. Frequency of actions for a specific region (window size: 20).

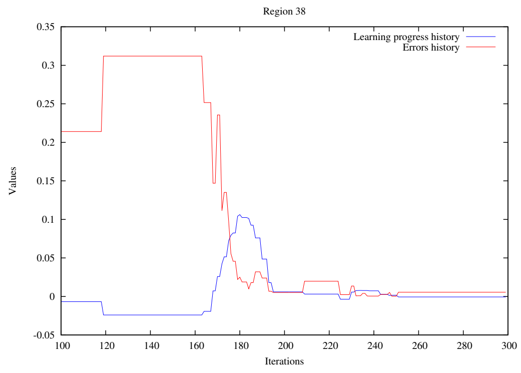


Figure 7. Learning progress increase for a certain region.

In Figures 8,9 the prediction ability of a learning machine over a sequence is illustrated.

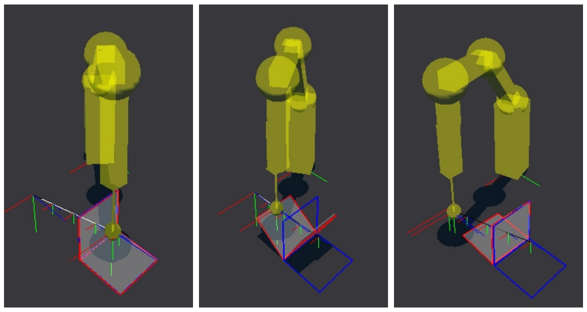


Figure 8. Prediction of the flipping affordance. The blue polyflap is the first predicted polyflap in the current sequence and the red one the last predicted one.

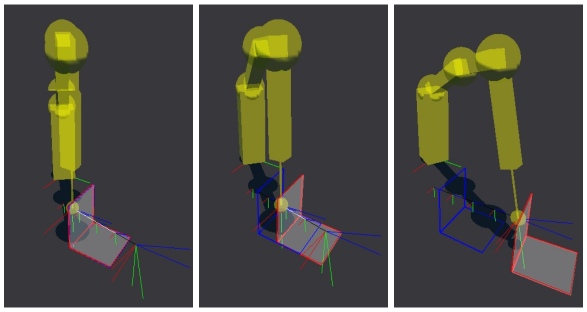


Figure 9. Prediction of the sliding affordance.

8 Conclusions and future work

The experiments shown in this paper demonstrate the ability of recurrent neural networks, in particular Long Short-Term Memory machines to approximate a regression function encoding the trajectory of simple geometrical objects when pushing actions are performed. Therefore, these machines are useful for predicting the affordances of pushing actions. We used 3-dimensional features and realistic simulations that we can then apply to real environments. Sequences of finger effector and polyflap poses were used to feed the LSTMs, showing the capacity of LSTM for prediction in relatively large time periods. The offline experiments showed great accuracy in prediction. The use of an active learning mechanism where machines are specialized in different parts of the sensorimotor space was also tested. The selection of actions is performed via a measure of learning progress that improves generalization.

In this work, the motivation to select an action via active learning is mainly based on the curiosity-driven mechanism introduced by the IAC algorithm. This mechanism forces the robot to select actions that maximize a learning progress measure. This encourages the reduction of error for sensorimotor regions that are still not accurately learned. The effectivity of the IAC-based strategy for assessing learning progress in sensorimotor regions with spatio-temporal features is confirmed. Moreover, machines that take into account spatio-temporal information fit well into the active learning loop. However, the gradient-based method for updating the networks still makes the process slow, so that many iterations are needed to observe high improvements. It is possible to add additional drives or measures for selecting actions in order to have different strategies for accelerating the learning progress. Additionally, alternative algorithms for LSTM training may also be considered.

The CrySSMEx[8] algorithm has been used to analyse recurrent networks as dynamical systems by using a conditional Entropy based method that extracts a probabilistic automaton associated to a machine. This method might be useful for active learning, because it represents uncertainty and predictability during the processing of spatio-temporal features.

REFERENCES

[1] Bram Bakker, ‘Reinforcement learning with long short-term memory’, in *Advances in Neural Information Processing Systems 14*, pp. 1475–1482. MIT Press, (2002).
[2] Paul Fitzpatrick, Giorgio Metta, Lorenzo Natale, Sajit Rao, and Giulio Sandini, ‘Learning about objects through action - initial steps towards artificial cognition’, in *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3140–3145, (2003).
[3] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins, ‘Learning to forget: Continual prediction with lstm’, *Neural Computation*, **12**, 2451–2471, (1999).
[4] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber, ‘Learning precise timing with lstm recurrent networks’, *J. Mach. Learn. Res.*, **3**, 115–143, (2003).
[5] J. J. Gibson, ‘The theory of affordances’, in *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, eds., R. Shaw and J. Bransford, 67–82, Lawrence Erlbaum, (1977).
[6] Alex Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, Ph.D. dissertation, Technische Universität München, July 2008.
[7] Sepp Hochreiter and Jurgen Schmidhuber, ‘Long short-term memory’, *Neural Computation*, 1735–1780, (1997).
[8] H. Jacobsson, ‘The crystallizing substochastic sequential machine extractor - `CrySSMEx`’, *Neural Computation*, **18**(9), 2211–2255, (2006).
[9] M. Kopicki, J. Wyatt, and R. Stolkin, ‘Prediction learning in robotic pushing manipulation’, in *Proceedings of the 14th IEEE International Conference on Advanced Robotics (ICAR 2009)*, Munich, Germany, (June 2009).
[10] P-Y. Oudeyer, F. Kaplan, and V. V. Hafner, ‘Intrinsic motivation systems for autonomous mental development’, *IEEE Transactions on Evolutionary Computation*, **11**(1), (2007).
[11] L. Paletta, G. Fritz, F. Kintzler, J. Irran, and G. Dorffner, ‘Learning to perceive affordances in a framework of developmental embodied cognition’, in *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pp. 110–115, (July 2007).
[12] B. Ridge, D. Skočaj, and A. Leonardis, ‘A system for learning basic object affordances using a self-organizing map’, in *International Conference on Cognitive Systems CogSys 2008*, Karlsruhe, Germany, (2008).
[13] Sergio Roa and Geert Jan Kruijff, ‘Long short-term memory for affordances learning’, in *Proceedings of the 9th International Conference on Epigenetic Robotics*, eds., Lola Caamero, Pierre-Yves Oudeyer, and Christian Balkenius, number 146 in Lund University Cognitive Studies, pp. 235–236. o.A., (11 2009).
[14] Sergio Roa, Geert Jan Kruijff, and Henrik Jacobsson, ‘Curiosity-driven acquisition of sensorimotor concepts using memory-based active learning’, in *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, pp. 665–670, (2008).
[15] *Towards Affordance-Based Robot Control - LNAI 4760*, eds., Erich Rome, Joachim Hertzberg, and Georg Dorffner, Lecture Notes in Computer Science LNAI 4760, Springer Verlag, Berlin, Germany, 2008.
[16] M. Schuster and K.K. Paliwal, ‘Bidirectional recurrent neural networks’, *Signal Processing, IEEE Transactions on*, **45**(11), 2673–2681, (Nov 1997).
[17] Aaron Sloman, ‘Polyflaps as a domain for perceiving, acting and learning in a 3-D world’, in *Position Papers for 2006 AAAI Fellows Symposium*, Menlo Park, CA, (2006). AAAI.

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Lisboa, Portugal, August 16 2010

Fitted Policy Search: Direct Policy Search using a Batch Reinforcement Learning Approach

Martino Migliavacca, Alessio Pecorino, Matteo Pirotta, Marcello Restelli and **Andrea Bonarini**[1]

Abstract. In this paper we address the combination of batch reinforcement-learning (BRL) techniques with direct policy search (DPS) algorithms in the context of robot learning. Batch value-based algorithms (such as fitted Q-iteration) have been proved to outperform online ones in many complex applications, but they share the same difficulties in solving problems with continuous action spaces, such as robotic ones. In these cases, actor-critic and DPS methods are preferable, since the optimization process is limited to a family of parameterized (usually smooth) policies. On the other hand, these methods (e.g., policy gradient and evolutionary methods) are generally very expensive, since finding the optimal parameterization may require to evaluate the performance of several policies, which in many real robotic applications is unfeasible or even dangerous. To overcome such problems, we propose the fitted policy search (FPS) approach, in which the expected return of any policy considered during the optimization process is evaluated offline (without resorting to the robot) by reusing the data collected in the initial exploration phase. In this way, it is possible to take the advantages of both BRL and DPS algorithms, thus achieving an effective learning approach to solve robotic problems. A balancing task on a real two-wheeled robotic pendulum is used to analyze the properties and evaluate the effectiveness of the proposed approach.

1 INTRODUCTION

This paper describes and evaluates a new Reinforcement Learning (RL) approach for robot learning that combines the advantages of two RL techniques often used in real-world applications.

Reinforcement Learning [26] is a set of well-established techniques which enable an agent to autonomously improve its behavior through direct interaction with the environment without having prior knowledge about the system dynamics and the task to be solved. Originally, RL techniques and theories have been focused on value-based temporal difference learning algorithms (of which Q-learning [31] is the most popular) for discrete domains. While acting in the environment, the agent updates a table which stores the estimates about the expected long-term utility of executing any action in each state. Since tabular approaches do not scale to high-dimensional or continuous-domain problems [26] (like robotic ones), many researchers have studied the use of function approximation to represent the value function. Although some successful applications have been reported (e.g., [28]), a lot of critical issues are involved like oscillations or even divergence of the approximator [2, 30].

These problems can be overcome by using *batch reinforcement-learning* (BRL) algorithms [9, 19] that can exploit the numerous regression methods from the machine learning literature while exhibiting strong convergence properties [1, 16]. By processing off-line and simultaneously all the information collected by the agent in the data acquisition phase, BRL algorithms have been used to solve complex continuous-state tasks [5, 24, 29] with usually much less data than online RL algorithms need. While BRL approaches have been proved very effective in continuous-state problems, the use of value-based methods, such as the fitted Q-iteration (FQI) [5], is generally ill-suited for learning in continuous-action domains, since the identification of the optimal policy requires to perform a maximization over the action space in order to find the action associated to the highest action value. Many approaches overcome this problem by simply considering a discretization of the action space, but this solution cannot be adopted in many applications where finer and smoother policies are needed. Furthermore, when the utility values associated to different actions are very close, the error induced by the use of function approximation often produces highly discontinuous policies which in robotic applications may have several negative effects, such as damages and high energy consumption.

On the other hand, *direct policy search* (DPS) methods [27, 18, 21] are well-suited to address continuous-action problems. Instead of estimating the action-value function over the whole domain, such methods define an optimization process over the policy space, where the goal is to find the policy which maximizes the expected return. Since a search over the whole policy space would be prohibitive even for simple problems, DPS methods generally restrict the search space by considering a fixed parameterized class of policies. The possibility of choosing the policy representation allows to supply the learning algorithm with prior knowledge about the domain or imposing constraints on the searched solution, thus strongly reducing (w.r.t. value-based approaches) the number of parameters being estimated by the learning process. In particular, these methods are well-suited for partially-observable domains since it is possible to design policies which depend only on the observable data from the underlying MDP [18]. For these reasons, DPS methods have been applied to a variety of robot learning problems [22, 13, 23]. The main drawback of DPS methods is that the performance evaluation of each different parameterization requires to repeatedly execute the corresponding policies, which, in robotic applications, can be a costly and time-consuming process, with a high risk of damages for the robot and the environment.

To overcome such drawback, in this paper we propose the fitted

[1] Department of Electronics and Information, Politecnico di Milano, piazza Leonardo da Vinci 32, I-20133, Milan, Italy, email: {bonarini,migliavacca,restelli}@elet.polimi.it, {alessio.pecorino,matteo.pirotta}@mail.polimi.it

policy search (FPS) approach, in which the performance evaluation step is run entirely offline using a modified version of the FQI algorithm to compute the value function of the specified policy. For each evaluation, FPS exploits the whole dataset of samples collected before the beginning of the learning process, without requiring to use the robot for gathering additional data about each evaluated policy.

The rest of the paper is organized as follows. In the next section, after introducing background material on MDPs, we present a brief overview of BRL and DPS methods and discuss their relevance for robotic applications. Section 3 introduces the fitted policy search approach. In Section 4, we evaluate its performance by making a comparative analysis with related learning algorithms on a real robotic problem: the balancing of a mobile two-wheeled pendulum. In the last section we draw conclusions and propose future research directions.

2 BACKGROUND

In this section we firstly introduce the basic RL notation and then we discuss the BRL and DPS techniques at the top of which our approach is built.

2.1 Markov Decision Problems

An RL problem can be defined as a Markov Decision Process (MDP) defined by a tuple $\langle S, A, P, R, \gamma \rangle$, where S is the continuous state space, A is the continuous action space, $P(s'|s,a)$ is the transition model that defines the transition density between state s and s' under action a, $R(s,a)$ is a reward function that specifies the instantaneous reward when taking action a in state s, and $\gamma \in [0,1)$ is a discount factor. The policy of an agent is characterized by a density distribution $\pi(a|s)$ that specifies the probability of taking action a in state s. The utility of taking action a in state s and following a policy π thereafter is formalized by the action-value function $Q^\pi(s,a) = E\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s = s_1, a = a_1, \pi\right]$, where $r_t = R(s_t, a_t)$. RL approaches aim at learning the policy that maximizes the action-value function in each state. The optimal action-value function is the solution of the Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \int_S P(s'|s,a) \max_{a'} Q^*(s',a') ds'.$$

The optimal policy is implicitly defined as the greedy policy $\pi^*(\cdot,\cdot)$, which takes in each state the action with the highest utility.

Temporal Difference algorithms [26] allow the computation of $Q^*(s,a)$ directly interacting with the environment with a trial-and-error process. Given the tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ (experience made by the agent), at each step, action values may be estimated by online algorithms, such as Q-learning [26], whose update rule is:

$$Q(s_t,a_t) \leftarrow (1-\alpha) Q(s_t,a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1},a') \right),$$

where $\alpha \in [0,1]$ is a learning rate.

2.2 Fitted Q-iteration

While in online learning the agent modifies step by step its control policy according to the experience gathered from the environment, the batch approach aims at determining the best control policy given a set of experience samples $D = \{\langle s_i, a_i, r_i, s'_i \rangle\}_{1 \le i \le N}$ previously collected by the agent according to a given sampling strategy. It is assumed that samples in D are enough to avoid conditioning problems in the regression algorithms [16]. In particular, good results have been achieved by *fitted Q-iteration* algorithms [5] derived from the *fitted value iteration* approach [8]. The idea of FQI is to reformulate the RL problem as a sequence of supervised learning problems.

Algorithm 1 FQI algorithm
```
  input: $D = \{\langle s_i, a_i, r_i, s'_i \rangle\}_{1 \le i \le N}$, a regression algorithm $\mathcal{R}$, number of iterations $L$
  initialize: $\hat{Q}_0(s,a) = 0, \forall s \in S, \forall a \in A$
  for $k = 1$ to $L$ do
    $T_k = \left\{ \left[ (s_i, a_i) \to r_i + \gamma \max_{a'} \hat{Q}_{k-1}(s'_i, a') \right]_{1 \le i \le N} \right\}$
    $\hat{Q}_k = \mathcal{R}(T_k)$
  end for
  return $\hat{\pi}^*_k(s) = arg \max_a \hat{Q}_k(s,a)$
```

Given the dataset D, in the first iteration of the algorithm, for each tuple $\langle s_i, a_i, r_i, s'_i \rangle$, the corresponding training pair is set to $(s_i, a_i) \to r_i$, and the goal is to use a regression algorithm to learn a function that approximates the expected immediate reward $Q_1(s,a) = E[R(s_t,a_t)|s_t = s, a_t = a]$. The second iteration, based on the approximation $\hat{Q}_1$ of the Q_1-function, extends the optimization horizon one step further, by estimating function $\hat{Q}_2$ through regression on the following training dataset: $T_2 = \left\{ \left[(s_i, a_i) \to r_i + \gamma \max_{a'} \hat{Q}_1(s'_i, a') \right]_{1 \le i \le N} \right\}$. By proceeding in the same way, at the i^{th} iteration, using the approximation of the Q_{i-1}-function, we can compute an approximation of the optimal action-value function at horizon i. The FQI algorithm is summarized in Algorithm 1.

The batch approach allows to use any regression algorithm, and not only parametric function approximators as happens for stochastic approximation algorithms. Several studies have reported very good results with a wide range of approximation techniques: kernel-based regressors [19], tree-based regressors [5], neural networks [24], CMAC [29], and advantage weighted regression [17]. All these works show how batch mode RL algorithms allow to effectively exploit the information contained in the collected samples, so that, even using small datasets, very good performances can be achieved. The size of the dataset is a key factor for robotic applications, since collecting a large amount of data with real robots may be expensive and dangerous. As shown in [24, 3], simple control problems, such as balancing a pole, can be solved from a dataset produced by taking random actions for a few minutes.

2.3 Direct Policy Search

For a long time, value-function algorithms have been the main approach to solve RL problems, achieving good results in many domains. Nonetheless, when more complex, real-world applications are considered the limitations of such approach come into play. The high sensibility to approximation errors, the computational cost in continuous-action problems, the generation of highly non-smooth policies, the impossibility of learning stochastic policies, and the difficulties in solving POMDPs are some of the reasons that make such methods unsuitable for real-world robotic applications.

Direct policy search methods partially overcome these limitations and for this reason have been largely adopted in continuous, high-dimensional, robotic domains [18, 22, 13]. Instead of estimating an action-value function which needs to be maximized over the action space to get the learned policy, DPS methods search a good policy within a restricted class of policies which are parameterized by some policy parameters θ: $\tilde{\Pi} = \{\pi_\theta : \theta \in \mathbb{R}^m\}$, where π_θ represents the policy $\pi(s, a, \theta)$. The learning process becomes an optimization process in the policy parameter space, where finding the optimal policy means to find the optimal parametrization θ^*, i.e., the parameterization that maximizes the expected return

$$J(\pi^\theta) = \int_S \mu_0(s) V^{\pi_\theta} ds = \int_S \mu_0(s) \int_A \pi_\theta(a|s) Q^{\pi_\theta}(s, a) da ds, \tag{1}$$

where μ_0 is a given probability measure over S.

Many different approaches have been studied, especially in the robot learning field, to optimize policy parameters. We can group all these methods into two main classes: *policy gradient* optimization and *global* optimization approaches. Policy gradient methods start from a given point in the parameter space and modify the parameter values following the steepest ascent on the expected return according to the gradient update rule

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J|_{\theta=\theta_k}.$$

Several methods have been proposed for obtaining a good estimation of the policy gradient $\nabla_\theta J|_{\theta=\theta_k}$ [20]. Due to the characteristics of the approach proposed in this paper (described in the next section), we limit our attention to one of the simplest and most used methods (in robotics) for gradient estimation: the finite-difference methods. The idea is to estimate gradients from roll-outs, by changing the current policy parameterization by small increments $\Delta\theta_i$. For each new parameterization the corresponding policy is evaluated and variations of the expected return are estimated with $\Delta\hat{J}_i = J(\theta_h + \Delta\theta_i) - J(\theta_h)$. The policy gradient estimate g_{FD} can be computed by regression of $\mathbf{\Delta\Theta} = [\Delta\theta_i]$ onto $\mathbf{\Delta\hat{J}} = [\Delta\hat{J}_i]$

$$g_{FD} = \left(\mathbf{\Delta\Theta}^{\mathbf{T}} \mathbf{\Delta\Theta}\right)^{-1} \mathbf{\Delta\Theta}^{\mathbf{T}} \mathbf{\Delta\hat{J}}.$$

The main advantages of finite-difference methods are the simplicity of the implementation, the possibility of using deterministic policies, and the possibility to learn without any knowledge about the model. On the other hand, such methods have many drawbacks. Each parameter has a different sensitivity to perturbations, thus requiring to define ad-hoc increments. Gradient estimate is negatively affect by perturbations associated to very bad policies. In stochastic systems the gradient estimate can be really noisy, requiring to perform many roll-outs to reduce estimation variance. To overcome these issues likelihood ratio methods have been proposed, the review of which is out of the scope of this paper.

Another issue that potentially affects any gradient-based optimization method is the risk of getting trapped in local maxima. To solve such problems global optimization methods (e.g., evolutionary algorithms [10, 11, 15] and cross-entropy methods [4]) have been often successfully used in DPS algorithms. The idea underlying these methods is to start with the generation of a number of candidate solutions from a prior distribution. On the basis of evaluations of these solutions, new candidate solutions replace the previous ones by preferring regions of the search space where good parameterizations were found, but giving a few chances also to less promising areas, with the purpose of escaping from local maxima.

The main drawback of all DPS methods is the cost of evaluating the expected return of the different parameterizations selected by the search algorithm, which in many simulated tasks is not an issue, but in real-world application can make the use of such methods unfeasible. Each evaluation requires to run several episodes from different

Algorithm 2 FPE algorithm for deterministic policies

```
input: $D = \{\langle s_i, a_i, r_i, s'_i \rangle\}_{1 \le i \le N}$, $E = \{s_i\}_{1 \le i \le N_E}$, a regression algorithm $\mathcal{R}$, number of iterations $L$, a deterministic policy $\pi : S \to A$
initialize: $\hat{Q}^\pi_0(s, a) = 0, \forall s \in S, \forall a \in A$
for $k = 1$ to $L$ do {Training}
    $T_k = \left\{ \left[ (s_i, a_i) \to r_i + \gamma \hat{Q}^\pi_{k-1}(s'_i, \pi(s'_i)) \right]_{1 \le i \le N} \right\}$
    $\hat{Q}^\pi_k = \mathcal{R}(T_k)$
end for
$\hat{J}_\pi = 0$
for all $s \in E$ do {Evaluation}
    $\hat{J}_\pi \leftarrow \hat{J}_\pi + \hat{Q}^\pi_L(s, \pi(s))$
end for
return $\hat{J}_\pi$
```

starting states, that can result in a very long process and, when a simulator is not available, it can use up the robot. In particular, it is well-known that gradient-based methods need to perform small update steps to the policy parameters in order to converge [25], thus requiring several policy evaluations. For what regards global optimization methods, besides the cost of performing several policy evaluations, there is also the risk of finding parameterizations that correspond to unstable or even dangerous policies, which can damage the robot and the environment.

In order to overcome these issues, we propose to avoid the direct evaluation of policies on robots by combining DPS methods with BRL algorithms thus obtaining the fitted policy search approach which will be introduced in the next section.

3 FITTED POLICY SEARCH

The main goal of the proposed approach is to perform the policy evaluation step without the need of executing several roll-outs for each new parameterization produced by DPS methods. The fitted policy search (FPS) approach uses at each iteration a set D of samples collected from the environment during an initial exploration phase as it happens in the FQI algorithm (see Section 2.2). Differently from FQI, FPS does not use these samples to compute the action-value function for the optimal policy, but to estimate the expected return $\hat{J}_\theta$ of the policy corresponding to the current parametrization (see Equation 1). For this reason, we introduce fitted policy evaluation (FPE), an on-policy version of the FQI algorithm, which, given a policy π performs an offline estimation of the action-value function $Q^\pi(s, a)$ using a set of samples $D = \{\langle s_i, a_i, r_i, s'_i \rangle\}_{1 \le i \le N}$. The difference with FQI is in the way the output values of the training set are built: at the k-th iteration, instead of maximizing the Q-function over the action space, for each sample i the training pair is defined as

$$(s_i, a_i) \to r_i + \gamma \int_A \pi(a|s'_i) \hat{Q}^\pi_{k-1}(s'_i, a) da.$$

Given the approximated action-value function $\hat{Q}^\pi$, the expected return for policy π can be estimated using Equation 1.

Since the integral over the action space is hard to compute for continuous, multi-dimensional action spaces[2], here we focus on the

[2] Other works have proposed approximated ways of estimating the value function even for stochastic policies without computing this integral. They define a regression problem based on samples collected by an exploration policy and weighted according to importance sampling [17, 14]

evaluation of deterministic policies $\pi : S \to A$, where the output values of each training sample can be easily computed as follows

$$(s_i, a_i) \to r_i + \gamma \hat{Q}^{\pi}_{k-1}(s'_i, \pi(s'_i)).$$

As a consequence, the expected return of policy π can be estimated by

$$\hat{J}(\pi) = \int_S \mu_0(s)\hat{Q}^{\pi}(s, \pi(s))ds. \qquad (2)$$

Since Q^{π} is obtained through regression methods, it could be represented using general (non-parametric) function approximators. For this reason, the integral in Equation 2 can be approximated using a Monte Carlo estimate over a set of states E sampled from the distribution of the initial state μ_0. The FPE algorithm for deterministic policies is summarized in Algorithm 2.

4 EXPERIMENTS

In this section we describe experiments performed on a traditional reinforcement-learning task: pole balancing. In particular, we apply our FPS approach to balance TiltOne, a real, mobile, two-wheeled robot designed and built in our robotics lab. The goal of the following experiments is to evaluate the effectiveness of the proposed approach in learning good policies exploiting only a very small set of data collected from the robot.

4.1 TiltOne

TiltOne is a two-wheel balancing robot that can stand in vertical position actuating only its wheels. The robot is $90cm$ tall and weights $20kg$ with batteries with a maximum payload of about $50kg$ when moving at speeds up to about $1.5m/s$. The wheels are $50cm$ in diameter and are actuated by two $150W$ DC motors with a $26:1$ gear reduction. A synchronous belt transmission adds a $4:1$ reduction, resulting in a total reduction of $104:1$ and a maximum continuous torque at the wheels of $18N/m$. The robot is controlled by an STM32 ARM Cortex-M3 microcontroller running at $72Mhz$, that is responsible of acquiring sensor outputs, controlling the motors, and logging the robot status communicating via a serial connection with an off-board computer.

The robot is equipped with sensors used to measure a set of variables that can describe the state of the system. The chosen variables are the angle ω of the frame with respect to the vertical position, the angular rate $\dot{\omega}$ and the speed of the robot on the horizontal plane $\dot{x}$. To evaluate the angle of the robot with respect to the direction of the gravity acceleration, an accelerometer and a rate gyroscope are used. The accelerometer is used to sense the gravity acceleration along two components, the first oriented from the wheel axis to the top of the robot frame and the other in the perpendicular direction. In static conditions, it is possible to evaluate the angle ω as

$$\omega = sin(g_x) \qquad (3)$$

where g_x is the horizontal component of the gravity acceleration in the frame coordinate system. In dynamic conditions, the acceleration of the robot is superimposed to the horizontal component of the gravity acceleration, making the measure too noisy for an accurate estimate of the angle.

The rate gyroscope is used to measure the angular rate with respect to the wheel axis. The sensor output is directly proportional to the angular rate, and in order to obtain the angle of the robot the signal needs to be integrated. This operation, as there is always a small offset in the signal, leads to drift problems, resulting in an increasing error over time.

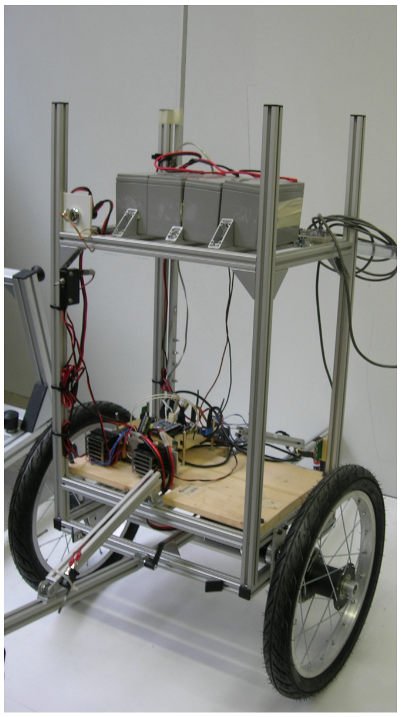


Figure 1. TiltOne

A Kalman filter is used to calculate a correct estimate of the angle, that is not affected by accelerations and drift problems, with the measurement gathered from the accelerometer and the gyroscope.

The speed of the robot is measured using two incremental optical encoders mounted directly on the motors, with a resolution of 2000 tics per revolution of the rotor.

4.2 Data acquisition

The datasets used by all algorithms are built on the basis of real data collected by the robot. The variable used are the angle of the robot ω, its angular rate $\dot{\omega}$, and the robot speed $\dot{x}$. A loop running on the microcontroller at $50Hz$ sends over a serial connection the instantaneous value of the three variables and the value of the control action currently performed.

In order to acquire data covering all the domain of the measured variables the acquisitions have been performed with the robot starting from the vertical position and controlled by actions taken uniformly at random. The value of the control is updated at a frequency of $10Hz$, so each action is maintained for 5 cycles to better evaluate the effect of a specific action on the robot behavior. When the robot falls (the absolute value of angle ω exceeds 8 degrees) the current acquisition is terminated and a new one is started. Depending on the random actions performed, a single acquisition can last from a fraction of second to about $2s$.

Each dataset is composed by 1000 points ($20s$ of acquired data) gathered from different data, to better evaluate the dependence of the performance observed from the train data used. From these complete datasets smaller ones have been extracted, with 100 and 500 points

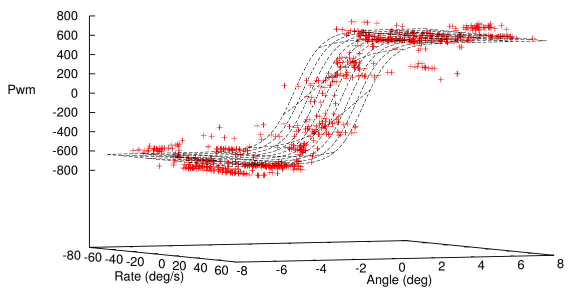


Figure 2. This graph displays the policy learned by FQI (projected on the $(\omega, \dot{\omega})$ plane) evaluated on a training dataset with $1,000$ samples (red crosses) and its approximation with the hyperbolic tangent function (black lines).

($2s$ and $5s$ of data respectively), to evaluate how the computed policy improves while increasing the dataset size.

4.3 Task Definition and FQI Parameters

To evaluate the performance of the proposed approach we have chosen the balancing task, so that the goal of the robot is to reach and keep the vertical position as soon as possible. To define such task we have chosen the following reward function:

$$r_{t+1} = -|\omega_{t+1}|.$$

Several tests have been performed to set all the learning parameters of the algorithms used in this experimental analysis. The time horizon was set to 10 undiscounted ($\gamma = 1$) steps both for the FQI and for policy evaluations. For function approximation we have used Extremely Randomized Trees [6], an algorithm for non-parametric regression that has been successfully used within the FQI approach in many reinforcement-learning applications [5]. Extremely randomized trees consist of an ensemble of regression trees that are built by choosing, in each node, the best splitting point (according to a variance reduction criterion) among a set of n_{cut} alternatives randomly generated. The splitting process stops when the number of samples in the node is smaller than the threshold n_{min} (whose value influences the degree of generalization and depends from the stochasticity of the system) and the node becomes a leaf. To each leaf is associated as value the mean of the outputs of its samples. Given an input, the corresponding output is computed as the arithmetic mean of the estimates produced by each tree. Following the indications of previous works [6] and the results of some tests, we used 50 trees generated by considering at each split step 4 (as the number of inputs of the Q-function) alternative cut directions. The minimum number of samples in the leaves was set to 10.

4.4 Policy Parameters

To choose the class of parameterized policies for this task, we ran the FQI algorithm on a dataset with $1,000$ samples to have an idea of the shape of the policy we were searching for. As expected, the policy estimated by FQI is quite discontinuous, but it is quite easy to see from Figure 2 that it can be well approximated by a hyperbolic tangent function of the angle. The other inputs have minor effects on the shape of the policy and seem to determine only slight translations. Starting from these observations we decided to consider the following parametric representation for the policy:

$$\pi^{\tanh}(\boldsymbol{\theta}) = \theta_1 \cdot \tanh\left(\frac{\omega}{\theta_2} + \frac{\dot{\omega}}{\theta_3}\right) + \theta_4 \dot{x}. \qquad (4)$$

It might be objected that, instead of performing a long optimization process involving several policy evaluations, the parameters $\boldsymbol{\theta}$ could be directly estimated by regression over the policy learned by FQI. Although, for certain problems, this approach may found effective policies, in the general case it does not work, even when the policy learned by FQI is actually the optimal one, since minimizing a distance in the policy space does not correspond to minimize the difference of their expected returns. In Section 4.6 we will experimentally show a case in which the direct fitting of the policy parameters over the FQI policy is clearly outperformed by the FPS approach.

4.5 DPS Methods and Learning Parameters

For the FPS approach we considered different DPS methods. Since the policy evaluation step can be easily performed for deterministic policies, we focus on DPS methods that search for such policies.

At first, we have tried the finite difference policy gradient method described in Section 2.3. The algorithm is initialized with a random parametrization $\boldsymbol{\theta}^0$. Each gradient estimation has been computed using 10 parameter perturbations $\Delta\theta_i \in [-50 : 50] \times [-0.2 : 0.2] \times [-10 : 10] \times [-0.1 : 0.1]$, and different learning rates have been considered for the parameter update. We observed that using high learning rates ($\alpha > 0.1$) the algorithm is not able to converge to a good parameterization, often exploring regions of the parameterization space associated to very unstable policies, where the gradient estimation becomes highly unreliable. Such phenomenon vanishes when very small learning rates are considered, but, as a result, a huge number of policies need to be evaluated, thus determining an excessive computational cost (even if it certainly requires less effort than directly evaluating the policies on the robot). These observations are confirmed by the results reported in [25] which compared the performance of different policy gradient methods on a simulated cart-pole problem. For the finite difference policy gradient approach, the authors found a good trade-off between performance and learning speed by setting the learning rate to 0.001, which results in more than $10,000$ gradient estimations before getting a near-optimal performance.

Since evaluating the performance of the finite-difference gradient method would have required too much time to get significant results, we decided to consider a simple greedy search method based on *random weight guessing* (RWG) [7]. As in the finite-difference method, at each iteration, new policy parameters are explored by considering K perturbations drawn uniformly from $[-50 : 50] \times [-0.2 : 0.2] \times [-10 : 10] \times [-0.1 : 0.1]$. If the best candidate policy outperforms the current solution, the latter is replaced and another iteration starts, otherwise the algorithm stops returning the current policy. In the following experiments, at each iteration, we have evaluated $K = 10$ candidate policies.

Finally, we have considered *evolutionary algorithms* (EAs), a well-known class of search and optimization strategies inspired by natural evolution theory, that have been often used to perform DPS for RL

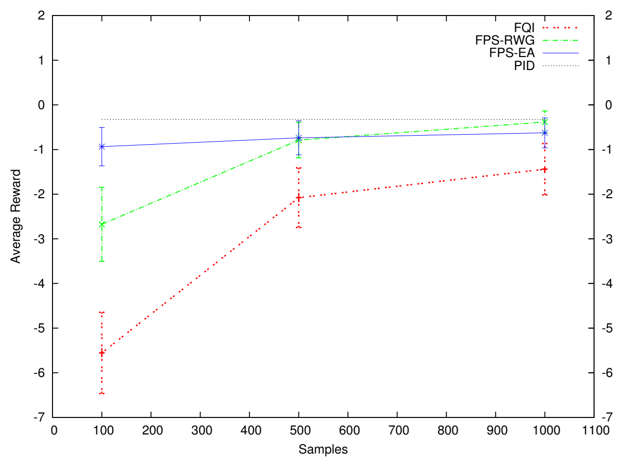


Figure 3. Comparison between FQI and two FPS variants (optimizing parameters of $\pi^{\tanh}$) on the TiltOne balancing task. The graph shows the average reward per step (computed on a $5s$ period) as a function of the number of samples in the dataset. Curves are an average over 25 runs. The horizontal dotted line corresponds to the optimal performance achieved by a carefully tuned PID controller.

tasks. On the other hand, the RL community has often snubbed the EA approach, starting from the assumption that a general optimization technique cannot compete with learning algorithms specifically designed to solve RL problems. In the following section, we will show that using EAs, and in particular genetic algorithms (GAs) [12], in the FPS approach allows to learn very good policies even with a very small set of samples. In our experiments, a tuning of the parameters of the algorithm led to evolve a population of 10 individuals for 10 generations, with a replace proportion of 0.9, a crossover probability of 0.5, and a mutation probability of 0.1.

4.6 Experimental Results

To estimate the effectiveness of the proposed approach, we analyze how the performances of two FPS algorithms (using RWG and EA as DPS methods) in the TiltOne balancing task change with respect to the size of the training dataset, and we compare them to the performance of the FQI algorithm. For each size of the training dataset (i.e., 100, 500, 1000 samples, representing $2s$, $5s$ and $20s$ of data acquisition), 5 independent acquisitions have been generated. After the initial acquisition phase, where TiltOne is used to collect datasets of transition samples, both FQI and the two FPS algorithms (which optimize over the parameters of the $\pi^{\tanh}$) carry out the whole learning process offline, without requiring additional samples. At the end of the learning process, the policies from the different algorithms are tested on TiltOne. For each policy, 5 testing episodes are performed and their results are averaged; each episode starts with the robot leaning on a support which determines an inclination $\omega = 2^o$ and stops after $5s$ (250 cycles). If the robot falls ($|\omega| > 8^o$) before the end of the run its reward is set to -10 for all the remaining cycles. The results are compared in Figure 3, where the performance is evaluated as the average reward per step. It can be observed that the FQI algorithm is not able to find a suitable policy with small datasets, as evidenced by the low average reward value. By increasing the length of the dataset the average performance improves, but with 1000 samples it is still far from the results obtained by the reference PID con-

	θ_1	θ_2	θ_3	θ_4	Average Reward
100	184.6	0.57	10.1	0.064	-3.28
	344.5	1.16	87.4	-0.006	-0.40
	671.6	1.45	29.1	0.313	-0.22
	100.0	0.19	50.8	-0.016	-9.26
	645.8	1.33	142.1	-0.021	-0.20
500	765.3	0.75	78.2	0.041	-0.34
	390.4	1.88	37.9	0.094	-0.65
	769.7	0.19	73.5	0.318	-2.54
	730.9	1.44	137.8	-0.006	-0.18
	765.4	1.93	95.8	0.011	-0.22
1000	787.1	0.58	49.4	-0.151	-1.21
	626.1	0.75	35.7	0.079	-0.18
	733.7	0.80	139.1	-0.073	-0.16
	755.4	0.93	71.6	-0.029	-0.14
	743.7	1.83	136.0	0.067	-0.23

Table 1. This table shows the parameters of $\pi^{\tanh}$ identified by FPS-RWG algorithm with different learning data and different dataset lengths, and the average reward on 5 runs

	θ_1	θ_2	θ_3	θ_4	Average Reward
100	637.0	1.82	105.4	0.037	-0.24
	750.1	0.57	133.7	-0.083	-0.43
	381.5	1.45	107.5	-0.074	-2.24
	278.8	0.53	129.4	-0.107	-0.99
	544.2	1.47	144.9	-0.237	-0.77
500	717.1	1.97	81.0	0.193	-0.29
	763.2	0.53	47.8	-0.048	-1.31
	781.0	0.36	41.7	-0.161	-1.73
	710.0	1.66	78.2	-0.059	-0.20
	637.8	1.01	93.3	0.140	-0.15
1000	750.3	1.32	69.6	0.034	-0.16
	694.6	1.35	58.4	0.078	-0.17
	604.0	1.09	137.5	-0.066	-0.18
	633.5	0.14	22.5	-0.201	-1.86
	727.8	0.52	68.2	0.092	-0.76

Table 2. This table shows the parameters of $\pi^{\tanh}$ identified by FPS-EA algorithm with different learning data and different dataset lengths, and the average reward on 5 runs

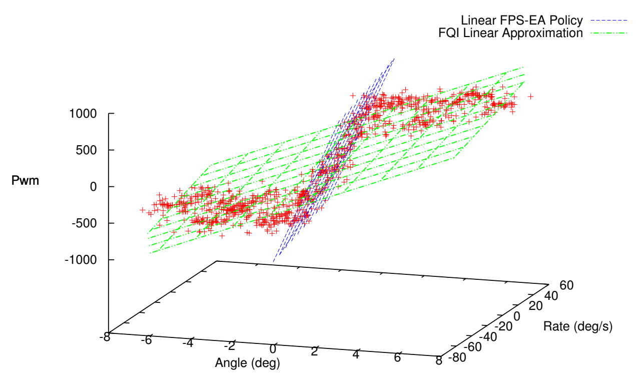


Figure 4. This graph displays the policy learned by FQI (projected on the $(\omega, \dot{\omega})$ plane) evaluated on a training dataset with $1,000$ samples (red crosses), its approximation using a linear function (green lines), and the linear function estimated using the FPS approach (blue lines).

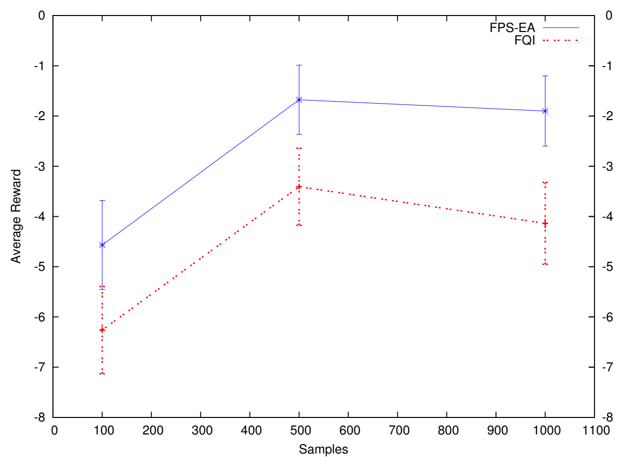


Figure 5. Comparison between FPS using EAs (optimizing parameters of π^{lin}) and the policy obtained by directly fitting π^{lin} over the FQI policy on the TiltOne balancing task. The graph shows the average reward per step (computed on a $5s$ period) as a function of the number of samples in the dataset. Curves are an average over 25 runs.

troller. On the other hand, FPS algorithms are able to show very good performance even with very small datasets, outperforming the results obtained with the FQI approach. The EA variant especially can learn good policies starting from as low as 100 samples ($2s$ of data) while the RWG variant, which is more prone to get stuck in local maxima, needs more samples to avoid the identification of bad parameterizations, while exploiting more samples both FPS variants perform closely to the reference PID controller. Although the number of samples is the main evaluation factor, it is interesting to compare the computational cost of the two FPS variants in terms of number of policy evaluations: the EA algorithm needs to evaluate a fixed number of policies (100 policies given by 10 individuals evolved for 10 generations), while the required evaluations using the RWG variant is highly dependent from the initial parameterization (we observed an average of 500 policy evaluations with a maximum of about 1500). As a consequence, in these settings FPS-RWG was much slower than FPS-EA in producing a policy. The parameters estimated by the FPS algorithms are shown in Table 1 and Table 2. Each row reports the parameters obtained from running the algorithms on different datasets and the average reward value (mean from 5 runs) achieved during the tests. It is possible to notice that, as expected, when the number of samples increases, the variability of the parameterizations returned by the FPS algorithms decreases.

The results observed by optimizing over the parameters of the $\pi^{\tanh}$ are close to what we expected, as the shape of the policy shown by the FQI approach reminds to $\tanh(\omega)$, so it seems the natural control law. For this reason, fitting this function directly on the policy learned by FQI leads to parameterizations and performance similar to those obtained by FPS algorithms.

To better highlight the difference between finding the parameterization that maximizes the expected return for a given family of functions and fitting that function to an existing policy, we defined a different parametric function π^{lin}:

$$\pi^{lin}(\boldsymbol{\theta}) = \theta_1 \cdot \omega + \theta_2 \cdot \dot{\omega} + \theta_3 \cdot \dot{x}. \tag{5}$$

Using this policy family leads to two clearly distinct results, shown in Figure 4: fitting it to the FQI policy simply finds the parameters that minimize the difference between the actions performed by the FQI policy and the parametric one, while the FPS algorithm optimizes the parameters in order to maximize the expected reward. Looking at the graph, it is easy to notice that the fitted function is globally closer to the FQI policy than the policy from FPS, but the actions around the equilibrium point ($\omega = 0$) are too soft for balancing the robot that will fall soon, getting a bad reward value. On the other side, the policy obtained from FPS methods is similar to $\pi^{\tanh}$ around $\omega = 0$, that is where the robot is supposed to operate, leading to a better expected reward value (see Figure 5 for performances and Tables 3 and 4 for the average parameter values).

	θ_1	θ_2	θ_3
100	79.4 ± 47.6	0.10 ± 2.70	-0.048 ± 0.151
500	127.2 ± 15.5	2.73 ± 1.27	0.012 ± 0.037
1000	126.2 ± 11.5	3.01 ± 1.52	0.013 ± 0.030

Table 3. This table shows the mean values and the standard deviations of the parameters of the linear function π^{lin} identified by directly fitting the linear function to FQI policy.

	θ_1	θ_2	θ_3
100	193.3 ± 98.8	1.15 ± 4.24	-0.147 ± 0.195
500	533.3 ± 217.9	13.36 ± 4.56	0.045 ± 0.239
1000	671.3 ± 178.1	5.92 ± 5.80	-0.126 ± 0.283

Table 4. This table shows the mean values and the standard deviations of the parameters of the linear function π^{lin} identified by the FPS-EA algorithm.

5 CONCLUSIONS

In this paper we have proposed fitted policy search, a direct policy search approach that, using a batch reinforcement learning algorithm, evaluates offline the expected return of explored policies, thus overcoming the main weaknesses of both BRL and DPS techniques when applied to robotic problems. Experiments performed with a two-wheeled balancing robot have shown that using DPS methods on a restricted class of smooth policies can outperform a value-based algorithm as FQI algorithms, leading to results comparable to the ones of the reference PID controller, even using a $2s$ long training dataset. Compared to other DPS approaches this method avoids to use the robot for evaluating a large number of policies, avoiding dangerous tests on the robot, and making the learning process feasible without the need of simulation tools.

Some aspects of the proposed approach will be improved in future works. Using a Monte Carlo approximation to compute the integral over the action space in the fitted policy evaluation algorithm, it will be possible to consider stochastic policies, thus opening the door to advanced policy gradient algorithms, such as natural gradients [14, 23]. In the near future, the proposed approach will be applied to more complex tasks, as controlling the balancing robot providing position and velocity set-points while maintaining its equilibrium.

REFERENCES

[1] A. Antos, R. Munos, and C. Szepesvari, 'Fitted q-iteration in continuous action-space mdps', in *Advances in Neural Information Processing Systems 20*, eds., J.C. Platt, D. Koller, Y. Singer, and S. Roweis, volume 20, 9–16, MIT Press, Cambridge, MA, (2008).

[2] L. Baird, 'Residual algorithms: Reinforcement learning with function approximation', in *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 30–37, (July 1995).

[3] A. Bonarini, C. Caccia, A. Lazaric, and M. Restelli, 'Batch reinforcement learning for controlling a mobile wheeled pendulum robot', in *Artificial Intelligence in Theory and Practice II: IFIP 20th World Computer Congress, TC 12: IFIP AI 2008 Stream, September 7-10, 2008, Milano, Italy*, volume 276, pp. 151–160. Springer Verlag, (July 2008).

[4] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuška, 'Policy search with cross-entropy optimization of basis functions', in *Proceedings 2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-09)*, p. 153160, (2009).

[5] D. Ernst, P. Geurts, and L. Wehenkel, 'Tree-based batch mode reinforcement learning', *Journal of Machine Learning Research*, **6**(1), 503–556, (April 2005).

[6] P. Geurts, D. Ernst, and L. Wehenkel, 'Extremely randomized trees', *Machine Learning*, **63**(1), 3–42, (2006).

[7] F. Gomez, J. Schmidhuber, and R. Miikkulainen, 'Efficient non-linear control through neuroevolution', in *Proceedings of the European Conference on Machine Learning (ECML-06, Berlin)*, volume 4212, pp. 654–662. Springer, (2006).

[8] G. J. Gordon, *Approximate Solutions to Markov Decision Processes*, Ph.D. dissertation, Carnegie Mellon University, June 1999.

[9] G.J. Gordon, 'Stable fitted reinforcement learning', in *Advances in Neural Information Processing Systems 8*, 1052–1058, MIT Press, (1996).

[10] V. Heidrich-Meisner and C. Igel, 'Evolution strategies for direct policy search', in *Parallel Problem Solving from Nature–PPSN X*, ed., G. Rudolph, pp. 428–437. Springer-Verlag, (2009).

[11] V. Heidrich-Meisner and C. Igel, 'Neuroevolution strategies for episodic reinforcement learning', *Journal of Algorithms*, **64**(4), 152–168, (2009).

[12] J.H. Holland, *Adaptation in natural and artificial systems*, MIT press Cambridge, MA, 1992.

[13] N. Kohl and P. Stone, 'Policy gradient reinforcement learning for fast quadrupedal locomotion', in *IEEE International Conference on Robotics and Automation*, volume 3, pp. 2619–2624, (2004).

[14] F. Melo and M. Lopes, 'Fitted natural actor-critic: A new algorithm for continuous state-action mdps', in *Proceedings of ECML Workshop on Principles and Practice of Knowledge Discovery in Databases*, volume 5212, pp. 66–81. Springer, (2008).

[15] D.E. Moriarty, A.C. Schultz, and J.J. Grefenstette, 'Evolutionary algorithms for reinforcement learning', *Journal of Artificial Intelligence Research*, **11**(1), 241–276, (1999).

[16] R. Munos and C. Szepesvári, 'Finite-time bounds for fitted value iteration', *Journal of Machine Learning Research*, **9**, 815–857, (2008).

[17] G. Neumann and J. Peters, 'Fitted q-iteration by advantage weighted regression', in *Advances in Neural Information Processing Systems 22 (NIPS 2008)*, 1177–1184, MIT Press, (2009).

[18] A.Y. Ng and M. Jordan, 'Pegasus: A policy search method for large mdps and pomdps', in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pp. 406–415, (2000).

[19] D. Ormoneit and Ś. Sen, 'Kernel-based reinforcement learning', *Machine Learning*, **49**(2), 161–178, (2002).

[20] J. Peters and S. Schaal, 'Policy gradient methods for robotics', in *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS 2006)*, pp. 2219–2225, (2006).

[21] J. Peters and S. Schaal, 'Reinforcement learning of motor skills with policy gradients', *Neural Networks*, **21**(4), 682–697, (2008).

[22] J. Peters, S. Vijayakumar, and S. Schaal, 'Reinforcement learning for humanoid robotics', in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp. 1–20, (2003).

[23] J. Peters, S. Vijayakumar, and S. Schaal, 'Natural actor-critic', in *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, volume 3720, pp. 280–291. Springer, (2005).

[24] M. Riedmiller, 'Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method', in *Proceedings of the Sixteenth European Conference on Machine Learning*, pp. 317–328. Springer, (2005).

[25] M. Riedmiller, J. Peters, and S. Schaal, 'Evaluation of policy gradient methods and variants on the cart-pole benchmark', in *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 07)*, pp. 254–261, (2007).

[26] R.S. Sutton and A.G. Barto, *Reinforcement learning*, MIT Press, Cambridge, MA, 1998.

[27] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour, 'Policy gradient methods for reinforcement learning with function approximation', in *Advances in Neural Information Processing Systems 12*, volume 12, 1057–1063, MIT Press, (2000).

[28] G. Tesauro, 'Td-gammon, a self-teaching backgammon program, achieves master-level play', *Neural computation*, **6**(2), 215–219, (March 1994).

[29] S. Timmer and M. Riedmiller, 'Fitted q-iteration with cmacs', in *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 07)*, pp. 1–8, (2007).

[30] JN Tsitsiklis and B. Van Roy, 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control*, **42**(5), 674–690, (May 1997).

[31] Chris Watkins and Peter Dayan, 'Q-learning', *Machine Learning*, **8**, 279–292, (May 1992).

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Lisboa, Portugal, August 16 2010

Using only aspects of interaction to solve shared attention

Renato Ramos da Silva[1] and **Roseli Aparecida Francelin Romero**[2]

Abstract. The communication between humans can be seen by the evolution of important cognitive functions and one of them is known as shared attention. This function usually is learned by humans in a period of childhood using basic set of structures and mechanism. In order to provide this learning ability in robots some approaches have been proposed. Models based on temporal difference, neural networks, probabilistic and reinforcement learning are examples used in several works. In our previous work [5], we have demonstrated how our robotic architecture enables a robot to learn the primary shared attention behavior. However, this needs some improvements to enable the development of others important parts in the construction of social robots. In this article, we have enhanced our robotic architecture, which is inspired on behavior analysis, to provide to the robot or agent, the capacity of sharing attention using only aspects of interaction. We have incorporated two improvements on ETG learning algorithms, named FAIETGQ and FAITEG, in it to simulate shared attention. Then, a set of empirical evaluations has been conducted in the social interactive simulator for performing the task of shared attention. The performance of this two algorithms have been compared with the Q-Learning algorithm, contingency learning algorithm and ETG algorithm. The experimental results show that the FAIETGQ algorithm required less memory capacity and presented better performance than other algorithms for shared attention problem.

1 INTRODUCTION

Infants, before near to complete one year, are able to follow another person's gaze to a location outside of their visual field: a key first step in establishing communication [31]. The communication between humans can be seen by the evolution of important cognitive functions and one of them is known as shared attention, also called "mutual gaze" by some authors [23].

The term shared attention is typically used to denote the attention of two people shared by looking at each other, rather than coordinating their attention on a third entity. While some authors, we use this assumption, use the terms joint and shared attention interchangeably to refer to the matching of one's focus of attention with that of another person, other authors make a distinction between them. Tomasello assigns joint attention as the attention provides by which two or more people coordinate and communicate their intentions, desires, emotions, beliefs, and/or knowledge about a third entity (e.g. an object or a common goal) [30].

But how does shared attention emerges at infants? One view explains the emergence of gaze following by postulating that infants gradually discover that monitoring their caregivers direction of gaze allows them to predict where interesting visual events will be [31]. Corkum and Moore demonstrated that 8-month-old infants can be trained to follow their caregivers gaze in a contingent reinforcement paradigm, where an interesting visual stimulus was shown if the infant followed the adults gaze to the stimulus location [4].

Triesch et all, also explains their basic set to construct an agent to learn shared attention and the approach emphasizes the role of biologically plausible choice of reinforcement learning [31]. This set includes perceptual skills and preferences, reinforcement learning, habituation and a structured social environment.

Reinforcement Learning (RL) [29] is a computational technique that allows an autonomous agent to learn a behavior via trial and error. The RRL task is very similar to the traditional RL task except that a relational representation is used for the states and actions [7]. Relational or first-order representation uses atoms, facts and relations to represent the environment.

This article reports an ongoing work aimed at developing our robotic architecture, which is inspired on Science of Behavior Analysis [2, 3, 17]. In order to provide this, we proposed two enhanced of ETG algorithm [5] by using only aspects of interaction. After this, we have incorporated five different learning algorithms in our robotic architecture. It has been inserted and evaluated in the simulator of social interactions. Then, the robotic architecture has been evaluated in the context of the shared attention.

This article is organized as it follows. We start with a brief background and related work section making the case for a rigorous experimental study of joint attention behaviors. After, we describe our robotic architecture, in which the learning mechanism will be inserted in, and social interaction simulator. In section 4, we present the basic concepts of this learning method and the main algorithms. Then, the main problem on development the reinforcement learning to resolve the shared attention are presented in section 5. The section that follows, we present the proposed FAIETGQ and FAIETG algorithms. Afterward, in section 7, the experimental results from a set of experiments carried out to evaluate the performance of the proposed architecture with each learning algorithm tested. A comparative analysis among the four learning algorithms is also been presented. Finally, in section 8, are presented the conclusions and future works.

2 BACKGROUND AND RELATED WORKS

The observable behaviors of an individual by attending to objects and events that others attend to can be regarding of "shared atten-

[1] Departamento de Ciência de Computação - Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo - Campus de São Carlos - Caixa Postal 668 - 13560-970 São Carlos, São Paulo, Brazil, email: ramos@icmc.usp.br
[2] Departamento de Ciência de Computação - Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo - Campus de São Carlos - Caixa Postal 668 - 13560-970 São Carlos, São Paulo, Brazil, email: rafrance@icmc.usp.br

tion" idea. In its simplest form, shared attention requires the establishment of eye contact between two people, followed by one person intentionally breaking it to be able to look at an object of interest, followed by a subsequent fixation of that object by the other person. In child-parent interaction, there is convincing evidence that even young children are sensitive to gaze and pointing cues and that they use those cues to follow the adult social partners attention, which facilitates early cognition and learning, according to developmental psychology [1, 12, 35].

At the end of the last century and the beginning of this can be considered as the starting point of studies of shared attention in the area of Human−Robot Interaction (HRI), particularly with Scassellati's works [24, 25]. Recently, several HRI studies evaluated shared attention behavior and we can divide in two groups. The basic difference into groups is that one is focused on interaction, where speech is used, and others on the learning method.

The first group we have found works using a story telling robot to find subjects were better able to recall the story when the robot looked at [13], a study where humans watched a video of a robot producing statements about a visual scene in front of it [34], a guide robot designed using data from human experiments to turn its head towards the audience at important points during its presentation [28].

Mutlu et al. [14] studied the extent to which eye-gaze behavior of the robot could signal participant roles (in a conversation) to human observers and confirmed that subjects' behaviors conformed to the communicated roles, and finally, investigates the temporal characteristics of shared attention processes in human-robot interactions in an experimentally unprecedented way [35].

Other group we have found works using a temporal-difference (TD) reinforcement learning scheme for learning joint visual attention [11]. Their model is limited in the sense that the infant only gets reward from the moving object operated by the observer. Also the caregiver's face is treated separately from the objects and does not lead to any reward.

Nagai et al. [16, 15] used face edge features and motion information (optical flow) to estimate the sensor motor coordination between these two inputs and the motor output using two separate neural networks. Their model does not utilize the depth information and thus can not handle ambiguous situations where an object appears in robots gaze direction that may not be located within the caregivers gaze direction.

Shon et al. [26] presented a probabilistic model of gaze imitation where estimated gaze vectors are used in conjunction with the saliency maps of the visual scenes to produce maximum a posteriori (MAP) estimates of objects looked at by the caregiver. A biologically plausible account of the development of gaze estimation skill is missing from their work.

Kim et al. [10] applied their model, based on basic set [31], on a robot head. They have been used an actor-critic reinforcement learning model for learning gaze following. They used map as additional information: a saliency map, representations of the caregiver head direction **h** and the caregiver eye direction **e**.

Finally, Silva et al. [5] have showed how shared attention can be resolved with reinforcement learning and relational representation. They have used ETG algorithm as learning mechanism in the robotic architecture. Here, the additional information used was called necessity, that represent an internal necessity of the robot. The basic idea of it was provided the ability of pro actively. But the same time it was used to help learn shared attention causing a large dependency.

3 ROBOTIC ARCHITECTURE AND SIMULATOR

The main objective of this chapter is to provide an basic overview of robotics architecture and social interaction simulator developed by us. More information can be found at [18, 21, 20, 6, 19, 5]

3.1 Our Robotic Architecture

The robotic architecture that is under development aims to build intelligent agent and it is based on Behavior Analysis theory[2, 3, 17]. This study is motivated to help in understanding the human being and help someone in many parts of the day to day, like robots assistants and entertainment activities. Thus, it is composed by three main modules: Stimulus Perception is State Estimation, Consequence Control is Motivation, and Response Emission Module is the Controller.

Figure 1 illustrates the general organization of the architecture and the interaction among the three main modules. Arrows indicate the flow of information in the three modules of the architecture. The circles indicate the methods and component structures of the modules.

The Stimulus Perception Module encodes stimulus from environment. Those stimulus are then used by the Consequence Control to verify internal necessity of robot and Response Emission Modules for learning and exhibition appropriate behaviors.

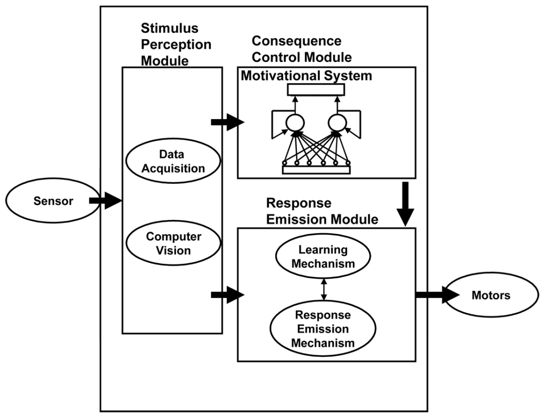


Figure 1. General organization of the architecture [21].

The Stimulus Perception Module may employ algorithms of data acquisition, a vision system, and a voice system, depending on the application domain. This module detects the state from the environment and encodes this state using an appropriate representation. The relational representation was chosen because it enabling the representation of large spaces in an economical way.

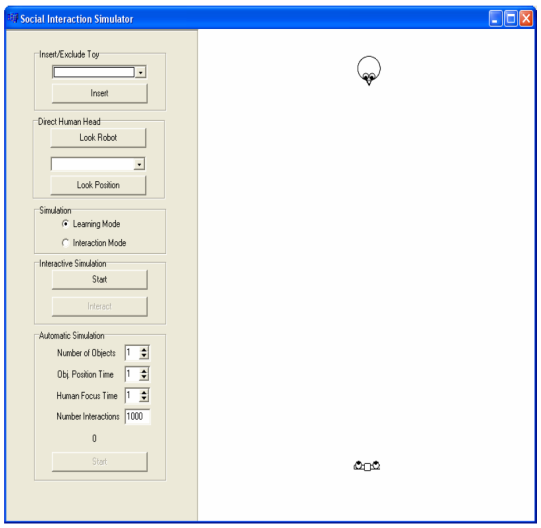
The Response Emission Module is composed by a learning mechanism that constructs a nondeterministic policy for response emission, that is, what response is to be emitted on the presence of certain antecedent stimulus. And the response emission mechanism that receives the information from learning mechanism and convert it in action to be executed by the motors.

The Consequence Control Module is composed by a motivational system modeled as a competitive artificial neural network with recurrent connections. It simulates internal necessities of the robot and generate a reward on the basis of the internal state estimate. An artificial motivational system may enable a robot to pro actively interact with the environment, driving its behaviors to satiate its artificial internal necessities. The motivational system is formed by necessity units that are implemented as a simple perceptron [9] with recurrent connections. A necessity unit simulates the internal necessities of an individual. A positive value of a necessity unit, above a predefined threshold, indicates the privation of the robot to certain reinforcement stimulus. In this way, the architecture supplies mechanisms to simulate privation states and satisfaction of necessities, and mechanism to determine reinforcements as consequences of an emitted response.

In the process of developing our robotics architecture new modules, such as emotions, mood and others, will be incorporated into. These new modules have direct influence in controlling the robot's necessity. A learning module that has a little or no dependence is a positive factor for the development of any robotics architecture developed for social robotics.

3.2 Social Interactive Simulator

To evaluate the proposed architecture, an interactive social simulator has been developed by us and it is presented here. It is based on works of Triesch and his colleagues [31]. This social interaction simulator is able to simulate an interaction between a robot and a human in a controlled social environment. Figure 2 shows the interface of the developed simulator. In this figure, on the left side of the interface is the control panel that enables interactive or automatic simulations, the human being is fixed on the upper side of the interface and the robot is fixed on the lower side of the interface.

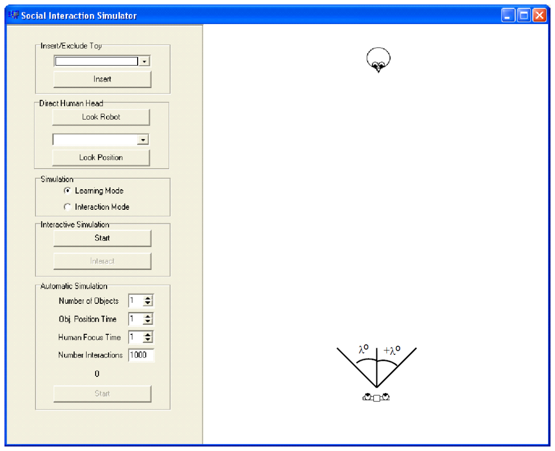


Number of Objects 1
Obj. Position Time 1
Human Focus Time 1
Number Interactions 1000
0

Figure 2. Social interactive simulator interface [21].

In order to simulate the shared attention task, it has been defined three entities that can be manipulated through functions of the simulator. They are a human, a robot, and two toys. The human being and the robot are positioned face to face, at a distance of approximately 50 cm from each other. The simulator enables that up two toys are positioned in the social environment. A toy can be positioned at any empty place of the social environment at any moment.

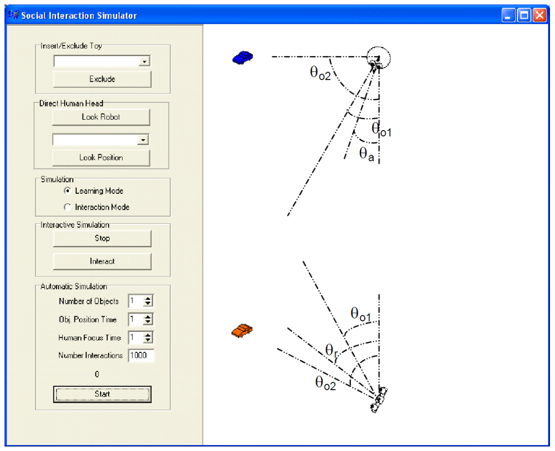
The social environment was modeled in the following way. Either the robot and the human can turn left or right their heads up to 90°. The robot has its central focus in 0° and has its visual field limited by a foveation parameter $\lambda°$, starting from the central focus, in $[-\lambda°, +\lambda°]$. Figure 3 shows the modeling of the visual field of the robot, in which lines represent the boundaries of the visual field.



Number of Objects 1
Obj. Position Time 1
Human Focus Time 1
Number Interactions 1000
0

Figure 3. Visual field of the robot [21].

The position of the robot's head is given by θ_r, that can assume values in $[-90°, +90°]$. The position of the human being's head is given by θ_a, that also can assume values in $[-90°, +90°]$. When an object i is positioned in the social environment, the simulator maps the angle between this object and the robot's focus, that is, the distance that the robot must move its head to focus the positioned object. This mapping is given by θ_{oi}, that can assume values in $[-90°, +90°]$. In this way, if an object is positioned in the environment, the simulator verifies if the same is inside the robot's field of vision, by comparing its position in relation to the robot's focus, considering the foveation of the robot. Figure 4 shows the positioning parameters of the robot, objects and human being, in which lines represent the distances between the robot'focus and objects as the position of the robot's head and human's head.



Number of Objects 1
Obj. Position Time 1
Human Focus Time 1
Number Interactions 1000
0

Figure 4. Positioning control [21].

Additionally, the simulator provides an adult attending stimulus that simulates attention from human being to the robot. The simulator provides the stimulus when the human and the robot are keeping eye contact and when the robot correctly follows the human gaze. This mechanism was incorporated in the simulator to validate the behavioral analysis presented by Dube and their colleagues [8], stating that the human serves as motivational operator in the context of shared attention learning.

During a simulation, the simulator executes interactions continually and each interaction takes 1 second. The simulator is able to position up to two simultaneous objects in the social environment, on places stochastically selected with probability ρ_o. These objects are positioned in the respective places for a time determined by the user (given in seconds). Additionally, the simulator is able to turn the human being's head to focus an object present in the environment or to focus the robot. The object that receives the human's focus is stochastically selected with probability ρ_{oi} and the human keeps his focus at the selected object for a time determined by the user (given in seconds), before turn his head to another object or to the robot.

4 RELATIONAL REINFORCEMENT LEARNING

In Reinforcement Learning (RL) [29], an agent navigates through an environment trying to earn rewards or avoid penalties. The environments state is described by a set of features, and the agent takes actions to cause the state to change. In one common form of RL called Q-learning [32], the agent learns a Q-function to estimate the value of taking an action from a state. An agents policy is typically to take the action with the highest Q-value in the current state, except for occasional exploratory actions. After taking the action and receiving some reward, the agent updates its Q-value estimates for the current state.

If we consider an agent navigator problem, we can represent the environment using a matrix with three dimension. Two of this to represent the space and one to store the Q-value for each action.

Relational reinforcement learning (RRL) has emerged in the machine learning community as a promising subfield of reinforcement learning (RL). It upgrades RL techniques by using relational representations for states, actions, and learned value-functions or policies to allow natural representations and abstractions of complex tasks [22].

The RRL task can be defined as it follows [7],
Given:

- A set of possible states S (represented in a relational format),
- a set of possible actions A (represented in a relational format),
- an unknown transition function δ: S x $A \rightarrow S$ (this function can be nondeterministic)
- a real-valued reward function r: S x $A \rightarrow \Re$.
- Background knowledge.

the aim is to find a policy for selecting actions π^*: $S \rightarrow A$ that maximizes a value function $V^\pi(s_t)$ for all $s_t \in S$.

Possible extensions of the (state, action) representation that can be added as background knowledge and it generally valid about the domain (states in S) can be specified in RRL [7].

RRL leads to a serious state space reduction. Then, the structure to represent the environment needs change to take advantage of this. Driessens [7] has cited three different approaches: decision trees called *TG* algorithm, kernel-based called *RIB* algorithm, or Gaussian Processes and kernels called *KBR* algorithm.

Other algorithm can be used ETG algorithm [27]. It is an enhancement of TG algorithm that was evaluated at the block word simulator. After, it was inserted in the robotic architecture and evaluated at the social interactive simulator [5].

The development of RRL algorithm to resolve the shared attention implies in, at least, one great problem.

5 THE PROBLEM OF SHARED ATENTION USING REINFORCEMENT LEARNING

One point when somebody start a project to development a social robot is the shared attention. The first perspective is to use only computer vision to resolve it, for example, we can calculate the angle that the caregiver turns and rotate it's head in other direction. It is a good idea, but the robot does not learn. This way put shared attention as innate reflex. According to psychology that behavior is learned in stages.

The choice of RL is related with biologically plausible of it [31]. Then, what are the difficulties to apply only RL on shared attention problem? In order to answer this question, we show through several steps as the main problem is hampering the implementation of it.

The Figure 5 shows the first stage of shared attention, when the robot establish eye contact with an human. In the figure the human is on top. Usually, the robot learn to do this with some interaction with the environment without objects and the human do not take action. The only positive reinforcement that the agent receives is to look at men. Then, it learns to give attention to a human.

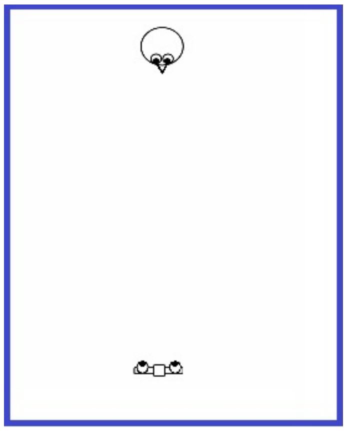


Figure 5. Eye contact.

After this, when the human has the attention of the robot one or more objects or events appear in the environment. The human look at it and the robot take a new state. The learning process the robot can use its knowledge or can explore the environment. It receive a positive reward if it looked at the same object, otherwise it receive a negative reward. Here, we adopt that the robot looked at the object. But the object can appear in several places. The Figure 6 has given us an example where the robot looked at a correct object in different places.

The moment when the robot looked at the object, in both cases(Figure 6 **a** or Figure 6 **b**), we have the same state for robot. At the end, the robot need establish eye contact again.

The great problem of use only RL algorithm to resolve shared attention is related with a large number of state meaning one state for robot. Then, the robot need a different action of the same state (robot view) to return establish eye contact. For example, in the Figure 6 **a**

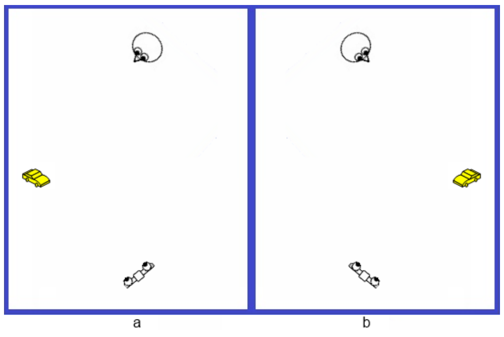


Figure 6. Look at the object.

the robot needs the action **X1** and in the Figure 6 **b**, in the robot view is the same, it needs a different action. This problem appear when the robot look at some object or when does not find it(negative reward) after had made eye contact.

In order to solve this problem, the algorithms based on RL have been used additional information. Kim used a Saliency Map, Caregiver Head Direction and Caregiver Eyes Direction from vision system module [10]. Otherwise, Silva used agent's necessity from motivational system module [5].

On the next chapter we present an improvement of this way to resolve shared attention.

6 ALGORITHMS PROPOSED

The dependency of the learning module from others has hindered the development of robotic architecture. We are looking for a solution so that we could only use aspects of interaction. Then, we found the possibility of using the previous action as a way of solving the problem.

The main idea is the fact that it is not necessary to worry about the moves that are made during the search, only with that action that resulted in the state that subsequently led to positive reinforcement. Using this purpose is that this work, we are proposing two enhanced of the current version of ETG algorithm [6].

The ETG algorithm learns a control policy for an agent as it moves through the environment and receives rewards for its actions. An agent perceives a state s_i, decides to take some action a_i, makes a transition from s_i to s_{i+1} and receives the reward r_i. The task of the agent is to maximize the total reward it gets while doing actions. Agents have to learn a policy which maps states into actions. Both enhancement follow this idea.

The learning mechanism takes from the environment state, an necessity of the agent, then it chooses (using the current policy) and takes an action. This process changes the state and the agent receives its reward. The reward can be either positive (equals to 10) or negative (equals to -1). After this occurred, the $qvalue$ is computed by:

$$\hat{Q}_i \leftarrow Q(s_i, a_i) + \alpha[r_{i+1} + \gamma \max(Q(s_{i+1}, a_{i+1})) - Q(s_i, a_i)] \quad (1)$$

Then, the relational regression engine receives a set of (state, action, $qvalue$, necessity) and tests the internal nodes if the state already exists. In the case this performance is false, the state is inserted in the tree and the leaf receives the action with the $qvalue$ and necessity values, forming a new branch. Otherwise, it updates the $qvalue$ for respective action in the leaf node.

In a leaf node, more than one action can be considered. For an easier access to the most adequate action, these actions can be ordered in decreasing order according to their $qvalue$ always that an example is inserted or updated. Each leaf also has a necessity associated with action and it refers to a necessity of the robot to choose this action on this state. Here, we use only the attention necessity. This process is repeated until there are not more interactions to be executed.

The great influence of necessity in the learning process hinders the development of other modules. The first change, we called as Free Additional Information ETG with Q-value(FAIETGQ), eliminates the interference of it and stores only positive examples in the tree. In a second step we try to verify the importance of the Q value in the learning process, then, we use all first changes but we removed the Q-value and we called it as Free Additional Information ETG (FAIETG).

In Algorithm FAIETGQ() is showed the processing of first enhancement ETG. The algorithm starts by initializing the Q-function and creates an empty regression tree [5].

Then, the set of (state, action, $qvalue$, last_action) is presented to relational regression engine. This process is repeated until there are not more interactions to be executed. All processing can be found in Algorithm FAIETGQ().

```
FAIETGQ()
BEGIN
  initialize the Q-function hypothesis(Q_0)
    and create a tree with a single leaf

  i = 0
  REPEAT
    take state s_i
    take action a_(i-1)
    choose a_i for s_i using a policy
     derived from the current hypothesis
     Q_i

    take action a_i, observe r_i and
     s_{i+1}
    IF(r_i positive){
      Update Q_i using the equation 1
      Update relational regression
       algorithm
        using (s_i, a_i, Q_i, a_(i-1))$ to
        produce Q_(i+1) // COMMENT{Use
         algorithm treeEngine()}
    }
    i = i+1
  UNTIL {no more interaction}
END
```

The relational regression engine receives a set of (state, action, $qvalue$, last_action) and tests the internal nodes if the state already exists. In the case this performance is false, the state is inserted in the tree and the leaf receives the action with the $qvalue$ and previous action (last_action) values, forming a new branch. Otherwise, it updates the $qvalue$ for respective action in the leaf node.

In a leaf node, more than one action can be considered. For an easier access to the most adequate action, these actions can be ordered in decreasing order according to their $qvalue$ always that an example is inserted or updated. Each leaf also has a previous action associated with action and it refers to action took by robot that result on this state. The tree algorithm adopted as a relational regression engine is presented in Algorithm treeEngine().

```
treeEngine()
BEGIN
  REPEAT
    sort the state down the tree using
```

```
     the tests of the internal nodes
     until to reach leaf node or null
    IF(the node is a leaf)
      IF(action exists)
        the Q-value is updated for the
         action in the leaf node
         according to the example
      ELSE
        the Q-value is inserted and the
         previous action for the action
         in the leaf node according to
         the example
      ENDIF
    ELSIF{the null is attained}
        generate a node
    ENDIF
  UNTIL{the example in a branch}
  IF{necessary}
    order actions in decreasing order
  ENDIF
END
```

At the second approach, we have followed all first changes and removed the Q-value. This proposal was used to evaluate the importance of Q-value for learning algorithm, because it is related mainly to create a better route to a goal.

```
FAIETG()
BEGIN
  initialize creating a tree with a
   single leaf

  i = 0
  REPEAT
    take state s_i
    take action a_(i-1)
    choose a_i for s_i using a policy
     derived from the current tree.

    take action a_i, observe r_i and
     s_{i+1}
    IF(r_i positive){
      Update relational regression
       algorithm using (s_i, a_i,
       a_(i-1)) // COMMENT{Use
       algorithm treeEngine2()}
    }
    i = i+1
  UNTIL {no more interaction}
END

treeEngine2()
BEGIN
  REPEAT
    sort the state down the tree using
     the tests of the internal nodes
     until to reach leaf node or null
    IF(the node is a leaf)
      IF(action not exists)
        the action with its previous
         action is inserted in the
         leaf node according to the
         example
      ENDIF
    ELSIF{the null is attained}
        generate a node
    ENDIF
  UNTIL{the example in a branch}
END
```

Besides the changes mentioned the new algorithm still have a further improvement in the first 100 rounds of learning. During this period, when the agent makes his search for the best action for present state, he does not choose that action has already been made earlier in which he received a negative reward.

Although the new algorithms allow the removal of the consequence control module, this will still remain because it is responsible for detects reinforcements received from the environment.

7 EXPERIMENT

In this section, the main results of the experiments carried out to evaluate the proposed learning algorithms are presented and discussed. The experiments were carried out employing the simulator previously presented, in the context of the emergence of shared attention. The purpose was to evaluate the capabilities of the new version of the robotic architecture on exhibit appropriate social behavior, learn from interaction, and generalize the learned behavior rules.

In the experiments three algorithms were utilized, in addition to previously proposed.Contingency Learning and RL, a version of Q-learning for our architecture that uses matrix, proposed by Policastro et al.[20]. Other algorithm is known to be precursors of the methods proposed in this paper, the ETG algorithm [5].

The experiments were composed by a learning phase of 10,000 time units (10,000 seconds in the simulator). During the learning phase, the human being initially kept the focus on the robot until it establish eye contact with him, characterized by 3 time units looking each other. Then, two objects were positioned in the environment and the human being turned his gaze for one of these objects, obeying the probabilities defined in the social interactive simulator. The human keeps his gaze at the object by 5 time units. Afterwards, the objects are then removed from the environment and the human turns his gaze to the robot, keeping the robot make eye contact again. This procedure is done in order to simulate a interaction where two agents are keeping eye contact and then one turns his gaze to an interesting event or object.

In the first 100 time units of learning phase, no objects are positioned in the environment and the human kept his focus on the robot the whole time, so the robot have learned that it may obtain the human attention by keeping eye contact with him. This procedure was done to shape the robot's behavior of looking for a human and keeping eye contact. After this, the learning phase was resumed using two objects as stated above. During the learning, the robot looks for the human. However, when an object is positioned in the environment and the human turns his gaze to it, the robot looses the human attention and starts to seek anything in the environment. Additionally, if the robot looks for a toy which the human this keeping his gaze, the human gives attention to the robot, in relation to the toy. In this way, after a history of reinforcement the robot will learn to follow the human's gaze to receive his attention and to satisfy its needs of socializing.

The learning capabilities of the architecture was analyzed by observing the robot interacting with the human and the environment, and computing a measure, the *correct gaze index* (CGI). The CGI measure is based on measures prosed by Whalen [33] and is defined as the frequency of gaze shifts from the human to the correct location where the human is looking at, given by:

$$CGI = \frac{\#shifts\ from\ the\ human\ to\ correct\ location}{\#shifts\ from\ the\ human\ to\ any\ location} \qquad (2)$$

To quantify the learning capabilities of the architecture through the learning of gaze following, at specific points during the learning process we temporarily interrupt the learning phase to evaluate its behavior. This evaluation was done by 10 runs of 500 time units (500 seconds in the simulator). For each run, the CGI value, given by Equation (2) was computed. After the evaluation phase, the learning process was resumed. A total of 20 interrupt points were placed. The whole procedure was performed 10 times and then a mean and standard deviation was calculated for each evaluation phase.

During the evaluation phase, the human initially kept the focus on the robot until it establishes eye contact with the human, characterized by 3 time units keeping eye contact. Then two objects were positioned in the environment and the human turned his gaze for one of these objects. However, in the evaluation phase, the object to which the human should turn his gaze was place on a position given by pre-established sequence (to prevent non determinism in the results). The second object (the distractor) was placed on an empty position, obeying the probabilities defined in the social interactive simulator. Once the robot turns its head to any direction, the simulator verifies if it is looking to the correct position in the environment (a toy which the human is looking for) or not, and update the CGI measure. This procedure takes 1 time unit. Afterwards, the objects are then removed from the environment and the human turns his gaze to the robot, keeping the robot make eye contact again.

For the experiments, the architecture knowledge was set as follows. Four stimulus were declared: *face*, *object*, *attention*, and *environment*, where attention is a reinforcer stimuli. Two facts were declared to define that red and blue objects are toys. Thirteen facts where declared in order to differentiate the human's head pose in frontal pose, six poses of left profile and six poses of right profile. Additionally, two more facts were declared to define when the robot is focusing the human or a toy.

In the response system was adjust the learning constant (α parameter) was set to 0.2. The discount factor (γ parameter) was set to 0.1. The exploration factor(ϵ parameter) was set to 0.05. Fourteen responses were defined so the robot can look at the human and search toys in six regions at left side and six regions at right side. This was done to divide the environment in regions of interest to make possible to follow the gaze to correct locations even in the presence of distractor toys.

The motivational system was set as follows. One necessity units were created: *socialize*. The activation threshold of the motivational system was set to 0.70. The sigmoid function inclination of the necessity unit (δ parameter) was set to 0.20. For the *socialize* unit, the Bias of the unit was set to 1.00 and its weight was set to 0.5, the weight of the recurrent connection was set to 1.00, weights of the input pattern (*hear(attention)*, *see(frontal(face))*, *see(toy(object))*, $see(looking - toy(object))$) was set respectively to -1.00, 0.05, 0.05 and 0.00.

When we are dealing with shared attention, a fact very important that it must be considered in all interactions is the number of times that the robot establishes eye contact with human. This is a essential fact for shared attention. The robot, through the simulator, can one of both options: to find anything in the environment or pay attention to human. If the robot choose only the first option, it could not simulate the shared attention. Because this, it is important that the learning algorithm maximizes the number of established eye contact.

The Figure 7 shows the average number of times that the robot establishes eye contact with human for each evaluation phase by using each one of the algorithms. In this figure, it is showed the beginning of the interaction between human and robot, in a total of 125 possible opportunities to establish eye contact each other.

The Figure 8 shows the same evaluation of the previous figure, but it considers the standard deviation for each evaluation phase.

In performing the analysis of the Figure 7 and Figure 8 we can verify that the new proposal FAIETGQ achieved a better final result than the other techniques. It still shows an increasing learning throughout the process, in addition to gradually reduce the standard deviation. This shows that it could learn to give attention to human. The graphics also show that the RL technique achieves a lower level of the other algorithms.

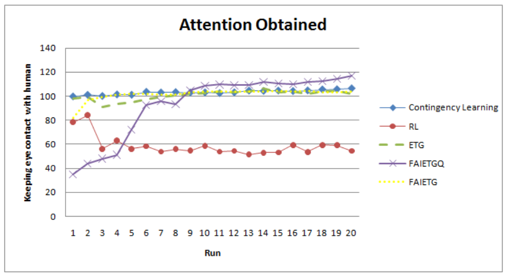


Figure 7. Average of attention obtained by human from the robot during evaluation phase.

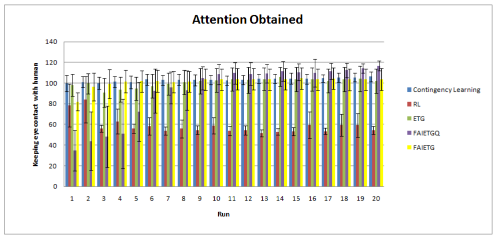


Figure 8. Average and standard deviation of attention obtained by human from the robot.

Figure 9 shows the performance, the learning progress over the time, of five different learning algorithms used as learning mechanism in our architecture to solve shared attention. It plots the CGI average value measured for each evaluation phase, at specific points during the learning process. Additionally, the Figure 9 shows the same the same evaluation with the standard deviation (indicated by error bars), for each run.

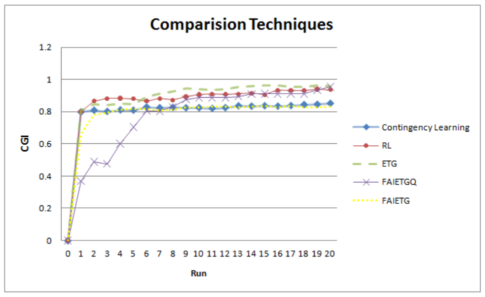


Figure 9. Learning evolution during the experiments.

Initially, it can be seen that all of the algorithms have not any knowledge about the problem. After the first run, all of them improve your knowledge attaining near of 80% of maximum CGI value, except the new proposals. In this stage, the robot or agent, learns a lot about the problem. After this, the contingency learning do not improve your knowledge until the end and this fact can be seen by your curve, remaining constant. Similarly the algorithm FAIETGQ starts like the others and gets an average of 60%. In the following section an evolution patarmar the same, and that does not change.

The ETG and RL algorithms have an slightly improve your learning after the first stage. Their curves are increasing over the time until to attain a stabilization level. FAIETGQ is different from others. It has a great and constant evaluation before first six runs and after it has a slightly improve until the end of the process. This tree algorithms have the same performance at the end of the performance.

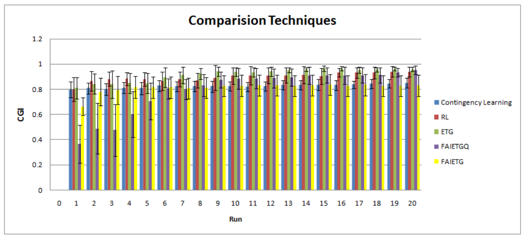


Figure 10. Attention Obtained by human of the robot.

A deeper analysis can be made considering Figures 7 and 9. Considering the factors of learning and the number of established eye contact, you can say that the FAIETGQ algorithm achieved better results. The experimental results showed that the contingency learning algorithm and FAIETG algorithm established a good number of eye contact than others, but it did not get a good CGI value compared to others. On the other hand, the RL established a lower number of eye contact than others, but it got a good CGI value. ETG presented a average value in both experiments, it is a good option. Overall, FAIETGQ has the better results with establish eye contact and take a good CGI result.

Another factor analysis is in relation to the memory requirements necessary for the algorithms: *Q-Learning* and ETG. The *Q-Learning* uses as a mechanism for knowledge representation, in this case, a *Q-Table* of 3,150 positions of memory. ETG algorithm used 68 nodes, with 29 leaf nodes in which 328 positions were occupied for the storage of actions, totalizing 396 positions of memory. However, FAIETGQ and FAIETG used 50 nodes, with 21 leaf nodes in which 63 positions were occupied for the storage of actions, totalizing 113 positions of memory. This demonstrates clearly that FAIETGQ algorithm required less memory than other algorithms.

The final analysis to be made is related to the two new proposed algorithms. The FAIETGQ algorithm obtained a better result than FAIETG. The diverging trends in learning shows that the first concepts learned are important for the FAITEG algorithm. With this result, we can infer that the manner in which the reinforcement is obtained can be improved.

8 CONCLUSION AND FUTURE WORKS

In this paper, we presented an ongoing work for the development of a robotic architecture inspired on Behavior Analysis. Five different learning algorithms, RL, contingency, ETG, FAIETGQ and FAIETG algorithm, were incorporated to robotic architecture to provide to the robot the ability of sharing attention. The learning mechanism were evaluated on a social interactive simulator and made by interacting real robotic head and the human in the context of the emergence of shared attention.

The experimental results show that the FAIETGQ, ETG and RL algorithms presented better performance than the contingency learning and FAIETG algorithm for shared attention problem. Another result is that FAIETGQ and FAIETGQ required less memory capacity than ETG and RL. Further, the results obtained show that the architecture is a potential tool to control sociable robots.

Future works include some improvements in the reinforcement generation and the extension of the architecture by implementing new mechanisms and skills like *verbal behavior*, *emotion*, *long term interaction control* and *learning by imitation*.

ACKNOWLEDGEMENTS

The authors would like to thank FAPESP, CNPq and CAPES for support received.

REFERENCES

[1] Chen Yu A, Dana H. Ballard B, and Richard N. Aslin C. The role of embodied intention in early lexical acquisition, 2004.
[2] A.C. Catania, *Learning, Interim (4th) Edition*, Sloan Publishing, 2006.
[3] John O. Cooper, Timothy E. Heron, and William L. Heward, *Applied Behavior Analysis (2nd Edition)*, Prentice Hall, 2007.
[4] Valerie Corkum and Chris Moore, ‘Origins of joint visual attention in infants’, *Developmental Psychology*, **34**(1), 28 – 38, (1998).
[5] Renato R. da Silva, Claudio A. Policastro, and Roseli A. F. Romero, ‘Relational reinforcement learning applied to shared attention’, in *IJCNN’09: Proceedings of the 2009 international joint conference on Neural Networks*, pp. 1074–1080, Piscataway, NJ, USA, (2009). IEEE Press.
[6] Renato R. da Silva, Claudio A. Policastro, Giovana Zuliani, Ednaldo Pizzolato, and Roseli A. F. Romero, ‘Concept learning by human tutelage for social robots’, *Learning and Nonlinear Models*, **6**(4), 44–67, (2008).
[7] Kurt Driessens, *Relational Reinforcement Learning*, Ph.D. dissertation, Katholieke Universiteit Leuven, Maio 2004.
[8] W. Dube, R.P.F. McDonald, R. Mansfield, W.L. Holcomb, and W.H. Ahearn, ‘Toward a behavioral analisys of joint attention’, *The Behavior Analyst*, **27**(2), 197——207, (2004).
[9] S. Haykin, *Neural Networks - A Comprehensive Foundation*, Prentice Hall, 1999.
[10] Hyundo Kim, H. Jasso, G. Deak, and J. Triesch, ‘A robotic model of the development of gaze following’, in *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pp. 238–243, (aug. 2008).
[11] G. Matsuda and T. Omori, ‘Learning of joint visual attention by reinforcement learning’, in *Int. Conf. on Cognitive Modeling (ICCM)*, (2001).
[12] Andrew N. Meltzoff, Patricia K. Kuhl, Javier Movellan, and Terrence J. Sejnowski, ‘Foundations for a New Science of Learning’, *Science*, **325**(5938), 284–288, (2009).
[13] Bilge Mutlu, Jessica K Hodgins, and Jodi Forlizzi, ‘A storytelling robot: Modeling and evaluation of human-like gaze behavior’, in *Proceedings of HUMANOIDS’06, 2006 IEEE-RAS International Conference on Humanoid Robots*, pp. 518 – 523. IEEE, (December 2006).
[14] Bilge Mutlu, Toshiyuki Shiwa, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita, ‘Footing in human-robot conversations: how robots might shape participant roles using gaze cues’, in *HRI ’09: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pp. 61–68, New York, NY, USA, (2009). ACM.
[15] Y. Nagai, ‘The role of motion information in learning human-robot joint attention’, in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2069–2074, (April 2005).
[16] Y. Nagai, A. Hosoda, and M. Asada, ‘A constructive model for the development of joint attention’, *Connection Science*, **15**(4), 211——229, (2003).
[17] W. David Pierce and Carl D. Cheney, *Learning, Interim (4th) Edition*, Psychology Press, 2008.
[18] C.A. Policastro, G. Zuliani, and R.A.F Romero, ‘Robotic architecture inspired on behavior analysis’, in *IEEE International Joint Conference on Neural Network, Orlando, Florida, USA*, pp. 1482——1487. IEEE, (2007).
[19] Claudio A. Policastro, Roseli A. F. Romero, Giovana Zuliani, and Ednaldo Pizzolato, ‘Learning of shared attention in sociable robotics’, *J. Algorithms*, **64**(4), 139–151, (2009).

[20] Claudio A. Policastro, Giovana Zuliani, Renato R. da Silva, Vitor R. Munhoz, and Roseli A. F. Romero, 'Hybrid knowledge representation applied to the learning of the shared attention', in *IJCNN*, pp. 1579–1584, (2008).

[21] Claudio Adriano Policastro, *Arquitetura rob?tica inspirada na an?lise do comportamento*, Ph.D. dissertation, Universidade de Sao Paulo, Outubro 2008.

[22] Marc Ponsen, Tom Croonenborghs, Karl Tuyls, Jan Ramon, and Kurt Driessens, 'Learning with whom to communicate using relational reinforcement learning', in *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1221–1222, Richland, SC, (2009). International Foundation for Autonomous Agents and Multiagent Systems.

[23] Elizabeth Redcay, David Dodell-Feder, Mark J. Pearrow, Penelope L. Mavros, Mario Kleiner, John D.E. Gabrieli, and Rebecca Saxe, 'Live face-to-face interaction during fmri: A new tool for social cognitive neuroscience', *NeuroImage*, **50**(4), 1639 – 1647, (2010).

[24] Brian Scassellati. Mechanisms of shared attention for a humanoid robot, 1996.

[25] Brian Scassellati, 'Imitation and mechanisms of joint attention: A developmental structure for building social skills on a humanoid robot', pp. 176–195. Springer-Verlag, (1999).

[26] A.P. Shon, D.B. Grimes, C.L. Baker, M.W. Hoffman, Shengli Zhou, and R.P.N. Rao, 'Probabilistic gaze imitation and saliency learning in a robotic head', in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2865–2870, (April 2005).

[27] R. R. Silva, C. A. Policastro, and R. A. F. Romero, 'An enhancement of relational reinforcement learning', in *IJCNN*, pp. 2055–2060, (2008).

[28] Maria Staudte and Matthew W. Crocker, 'Visual attention in spoken human-robot interaction', in *HRI '09: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pp. 77–84, New York, NY, USA, (2009). ACM.

[29] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[30] Michael Tomasello, Malinda Carpenter, Josep Call, Tanya Behne, and Henrike Moll, 'Understanding and sharing intentions: The origins of cultural cognition', *Behavioral and Brain Sciences*, **28**, 675–735, (2005).

[31] Jochen Triesch, Christof Teuscher, Gedeon O. Dek, and Eric Carlson, 'Gaze following: why (not) learn it?', *Developmental Science*, **9**(2), 125 – 147, (2006).

[32] C J C H Watkins, *Learning from Delayed Rewards*, Ph.D. dissertation, University of Cambridge, 1989.

[33] C. Whalen and L. Schreibman, 'Joint attention training for children with autism using behavior modification procedures', *Journal of Child Psychology and Psychiatry*, **44**(3), 456––468, (2003).

[34] Akiko Yamazaki, Keiichi Yamazaki, Yoshinori Kuno, Matthew Burdelski, Michie Kawashima, and Hideaki Kuzuoka, 'Precision timing in human-robot interaction: coordination of head movement and utterance', in *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 131–140, New York, NY, USA, (2008). ACM.

[35] Chen Yu, Matthias Scheutz, and Paul Schermerhorn, 'Investigating multimodal real-time patterns of joint attention in an hri word learning task', in *HRI '10: Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction*, pp. 309–316, New York, NY, USA, (2010). ACM.

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Lisboa, Portugal, August 16 2010

Sub-Rationality and Cognitive Driven Cooperation

José Ferreira de Castro[1]

Abstract. The M-Logic Machine is a proposal for a Unified Theory of Cognition [1][2][4][11][12][13] that uses cinematic memories for learning from scratch in a sensory-motor setting. Its choices are sub-rational in the sense that they depend on coarse pleasure/pain evaluations that do not search to maximize expected rewards. This simple cognitive approach is applied to the Prisoners Dilemma, clarifying the necessary conditions for cooperation.

1 INTRODUCTION

One of the most interesting subjects when we consider autonomous agents is the emergence of cooperative behaviour. To achieve it, the importance of evolutionary learning seems obvious. But it is also obvious that our cognitive abilities play an important part in the decision to cooperate or defect. In AI a rational agent is often defined as an agent that is able to solve optimization problems, where some value functions are to be optimized. In a competitive world with limited resources this often favours the most greedy and selfish strategies. Optimization also requires some form of omniscience, i.e. a complete knowledge of the alternative strategies and their consequences. This short paper presents an original alternative, the M-Logic Machine (MLM) that does not assume any type of omniscience and does not seek optimization. It explains how the decision mechanisms of the MLM lead to cooperation under certain conditions that are in full agreement with our intuition.

2 THE M-LOGIC MACHINE

2.1 Omniscient Machines

Let us first consider a simple decision model where several states of reality $\phi_1, \phi_2, \phi_3 ... \phi_{\rm n}$ arise with a known probability distribution $p_1, p_2, p_3 ... p_{\rm n}$. Confronted with this set of possible states, the agent can choose among a set of available different actions $a_1, a_2, a_3 ... a_{\rm m}$ that will ultimately generate a known reward value $v_{\rm ij}$ for each specific state and action. A decision table is thus defined:

Table 1. Omniscient Decision Table

	p_1 ϕ_1	p_2 ϕ_2	p_3 ϕ_3	... ...	$p_{\rm n}$ $\phi_{\rm n}$
a_1	v_{11}	v_{12}	v_{13}	...	$v_{1{\rm n}}$
a_2	v_{21}	v_{22}	v_{23}	...	$v_{2{\rm n}}$
a_3	v_{31}	v_{32}	v_{33}	...	$v_{3{\rm n}}$
...	...	...	...	...	...
$a_{\rm m}$	$v_{{\rm m}1}$	$v_{{\rm m}2}$	$v_{{\rm m}3}$	...	$v_{\rm mn}$

[1] CENTRIA, Universidade Nova de Lisboa Portugal, email: CastroJFGF@gmail.com

The obvious rational strategy is to choose the action a that maximizes the expected reward, given the known probability distribution of the states. The learning problem here is to ascertain this omniscient decision table. In that table the states are permanent entities that encompass the past, the present, and the future. If the states result from a constant environment with known fixed laws of nature, the permanent probability distribution can in principle be estimated from a finite time slice. In other words, a stationary process is assumed. But in a multi-agent setting, with changing populations of agents interacting with each other, the actions taken will feed back to the temporal probability distribution, often without stable winning strategies. In that case we are driven to the game theory of iterated games, further complicated by the fact that the assumption of a stationary process may no longer be realistic. These issues will be further considered in section 3. In any case, an efficient autonomous machine should learn a good decision table by itself, and learn it fast enough, because a realistic machine, even dealing with a stationary process, can only spend a limited amount of energy and has limited resistance to structural damage. A good enough decision table must be learned in time to assure the survival of the machine. Any real machine has also a limited range of sensory abilities that must cope with noise and irrelevant information. I argue that these types of constraints are better handled when we replace omniscient states by small cinematic descriptions of reality, captured by the available sensors and pre-processed in order to become simple and useful. A *unified theory of cognition*, not just some clever learning algorithm, is required to address the many problems involved in cinematic learning.

2.2 MLM: A Sub-Rational Machine

The M-logic Machine (MLM) has been developed by the author since 2001, and its current source code can be found on the authors homepage (https://sites.google.com/site/josefgfcastro/). The MLM learns from scratch in a sensory-motor setting, and its basic features can be subject to evolutionary learning. We now explain the rationale behind the MLM architecture and its basic assumptions. The state of a M-Logic Machine M at time step t is the state of all its internal memories and sensors, and is noted $\Omega_M(t)$. In order to build a frugal decision table for a MLM machine evolving in time, let us consider a list S of short scenes built from the machine's measurements, $[s_{\rm p} ... s_3, s_2, s_1]$. In current MLM implementations, the number of scenes in S is limited to two hundred for long-term memories. A list of scenes with the most recent scene at the head of the list and with the remaining scenes placed in chronological order is called a *tale*. Some *recording criteria* are used to start or stop recording a new scene into a tale. New scenes are inserted in the top of the list, and the bottom elements that are pushed beyond the size limit are erased. This limiting mechanism is used in all the lists of the MLM. We shall call these lists *bounded lists*. Each scene s in S is a cinematic record of the measurements performed by the machine, represented by a list of frames $[f_{\mathrm{q}}...f_3, f_2, f_1]$. The most recent frame f_{q} is at the head of the list. Each frame in a scene records a small set of simultaneous measurements. Each different possible set is called a *sensory mode*. In each scene the sensory mode is kept constant. Sensory modes select only a few of the available measurements, according to the orienting reflexes of the machine [9][10]. This limitation to a few measurements is needed for the MLM to work, and agrees with what happens in natural thought [3]. Each frame f is represented by a fixed-lenght list of channels (a vector) $[c_1, c_2, c_3...c_{\mathrm{r}}]$. The sensory mode assigns each selected measurement to a specific channel. At each time step, the recording status $\rho(\Omega_M(t))$ and the current sensory mode $\mu(\Omega_M(t))$ tell the MLM whether to record or not and what to record (and where in the frames), respectively. We start assuming that all sensory modes include measurements of the immediate rewards and the actions taken at each moment. The MLM builds its scenes as sequences of two micro-steps: first the inner and outside world situation is measured (this includes the immediate rewards resulting from the previous steps), leading to an internal choice and triggering of voluntary actions, and then the immediate actions are measured. This is similar to what happens in living beings, where muscle contractions are triggered before they are detected by muscle neurons. The decision process is not measured. We trigger our voluntary actions before actually knowing what we triggered. We shall therefore rewrite the frames in a scene as $[(f_A/f_S)_{\mathrm{q}}...(f_A/f_S)_3, (f_A/f_S)_2, (f_A/f_S)_1]$, where each f_A measures and records the motor actions performed by the machine after f_S measures and records the world situation and the immediate rewards. This micro-step pattern brings natural constraints to the pace of frame generation. A scene $[(f_S)_{\mathrm{q}}...(f_A/f_S)_3, (f_A/f_S)_2, (f_A/f_S)_1]$ with only f_S in its head in noted s^*. Besides, any scene s may be written as a string of sub-scenes $s_{\mathrm{a}}s_{\mathrm{b}}s_{\mathrm{c}}...s_{\mathrm{k}}$. One natural way to split a recorded scene into relevant sub-scenes is to anchor one of its frames to the present moment t_{p}. The generic split $s_\beta s_\alpha^*$ is defined as follows: given a frame anchored to the present-moment $(f_A/f_S)_{\mathrm{i}}^{t_{\mathrm{p}}}$, all frames before the anchored frame are included in s_α^*, and all frames past the anchored frame are included in s_β . The anchored frame $(f_A/f_S)_{\mathrm{i}}^{t_{\mathrm{p}}}$ is split in two: f_S (the most recent past state) is included in s_α^* and f_A (the nearest future instant action) is included in s_β. The nearest future immediate reward is the one found in the frame that follows the nearest future instant action frame. Let us assume that the scenes obtained from reality are being recorded and stored in a long-term memory we call *Dominance List Memory* (DLM). In other words, DLM is a bounded list of short scenes $[s_{\mathrm{p}}...s_3, s_2, s_1]$ recorded by the machine as it evolves in its environment. It's not a tale in the sense that the listed scenes may be eventually reordered by a dominance process. For a given time step t the list of short scenes that are found in DLM is noted DLM(t). In each DLM only a few measurements are placed in the frames according to a constant sensory mode μ, therefore the scenes list will be written DLM(t, μ). My proposal is to replace the states in the omniscient decision table by s_β sub-scenes extracted from DLM(t, μ) by an interrogation template $I(s_\beta)$. Let us suppose that the machine is currently recording a scene in a short-term memory STM(t_{p}, μ) that keeps track of the recent past (up to twenty frames) until the present-time t_{p}. A sub-scene s_α^* (up to three frames) is taken from STM(t_{p}, μ) before an action is triggered, and then DLM(t, μ) is searched for sub-scenes that match the required context $C(s_\alpha^*)$ associated to the interrogation template $I(s_\beta)$. The s_β found are possible continuations for the s_α^* scene. Each s_β provides a sequence of actions starting at t_{p}. We can use the $s_\beta s_\alpha^*$ scenes to replace the ϕ states in the Omniscient Decision Table. This radically changes the nature of the decision table, because the sequence of instant actions to be taken and the corresponding instant rewards are defined by the $s_\beta s_\alpha^*$ scenes that were found in DLM(t, μ). With this approach, the decision table becomes a *Sensory Decision Table* (see Table 2 hereafter), because all data in the table are simply what are given by the machine's sensors, rather than given a priori by an omniscient intelligence. The Sensory Decision Table no longer relies on background knowledge that models a certain domain, although it still relies on the machine's ability to measure immediate rewards. The uncertainties of the sensors relative to the omniscient model are not handled by the in-life cinematic reinforcement learning process. Agents with sensors and orienting responses that are not good enough for survival will simply extinguish. These aspects will be handled by evolutionary learning. In the left upper corner of the Sensory Decision Table are two elements: the source of information DLM(t, μ) and the context-interrogation query $C(s_\alpha^*)I(s_\beta)$ used to find the different sub-scenes s_β in DLM(t, μ) that are possible continuations to s_α^*. The number n of recorded continuations will be in general larger than the number m of the corresponding recorded action continuations (i.e. $n > m$). It is true that each s_β^{i} defines a single u_{i} (a utility value calculated from the instant rewards recorded in s_β^{i}) and a single a_{i} (the series of instant actions in s_β^{i}). Therefore for each column there is only one utility value defined. But different states and utilities may relate to the same measured action sequence a_{i}, either because the modeled reality is inherently probabilistic, or due to the imprecision of the measurements. The probability p_{i} of each continuation s_β^{i} can be easily estimated by counting for each i the number of identical continuations s_β^{i} found in DLM(t, μ). The Sensory Decision Table can therefore be learned on-the-fly by the machine. It is built as the machine acts on itself and its environment, first at random, and then using the table to choose the actions with the best known expected utility. The configuration of the Sensory Decision Table built at a given time t on a sensory mode μ is noted SDT(t, μ). Because the DLMs are bounded lists, the size of the corresponding SDTs can actually decrease. If the MLM finds a best a_{i}, the other continuations in DLM will be eventually pushed to oblivion.

Table 2. Sensory Decision Table

DLM(t, μ) $C(s_\alpha^*)I(s_\beta)$	p_1 s_β^1	p_2 s_β^2	p_3 s_β^3	... ...	p_{n} s_β^{n}
a_1	u_{11}			...	
a_2		u_{22}		...	
a_3			u_{33}	...	
...	...	...	...	...	...
a_{m}				...	u_{mn}

We can choose a single-framed interrogation template $I(s_\beta)$. This choice brings up a greedy and short-sighted machine that will not look for greater rewards beyond immediate losses. Increasing the number of frames in the $I(s_\beta)$ template allows an evaluation of the utility over a longer time span. But it is not possible to know from the start the final rewards found in the Omniscient Decision Table.

As before, given a Sensory Decision Table SDT(t, μ), a still plausible rational choice is the action that maximizes the known expected rewards (i.e. those that were already measured and recorded). But now rationality is clearly bounded by the finite amount of knowledge available to the machine. Its knowledge starts from nothing and grows up to a certain DLM size limit that is typically too small to record all the possible s_β for the possible s^*_α. The information available in the Sensory Decision Table is typically so scarce that the evaluation of a probability distribution for the s^{i}_β can be grossly misleading: too few instances of each s^{i}_β are found in DLM to correctly evaluate the corresponding p_{i}. Also there can often be a total lack of information in the Sensory Decision Table for some action a. This brings the usual dilemmas between exploitation and exploration. Suppose all the known utilities are negative, but the Sensory Decision Table does not cover all the possible actions. Should the machine choose the least negative expected value deduced from the available data, trying to minimize its losses, or should it rather risk some action not yet performed? Heuristics and meta-learning abilities are needed to address these issues.

Given the limitations of the Sensory Decision Table, the M-Logic Machine goes a step further. It gives up the idea of evaluating the probability distribution. Instead, it takes action based on the first s_β found in the DLM. No decision table is used, just a scan of the DLM scenes. All scans start from the head of the lists, and stop after a first match is found. For this solution to work, two things are required. First, some aspects of s_β must be included in the query context C (not just s^*_α). The context-interrogation query becomes $C(s^*_\alpha, s^c_\beta)I(s^i_\beta)$. The split of s_β among the context C and the interrogation I is determined by the current heuristic h. Second, the DLM must be sorted in such a way that the more accurate predictions are found first. To achieve predictive accuracy, the M-Logic Machine constantly reorders the DLM list of scenes, pushing down in the DLM the scenes that offer wrong predictions and pulling up in the DLM the scenes that offer correct predictions. This is called *the updating dominance mechanism*.

The context-interrogation query $C(s^*_\alpha, s^c_\beta)I(s^i_\beta)$ includes a required future s^c_β. Any subset of the current sensory mode can be assigned to s^c_β, but the most obvious choices are the measurements related to the current survival needs of the machine, either the ones that need to be found or the ones that need to be avoided. Smart orienting reflexes are needed to trigger the adequate sensory modes. In a sensory-motor setting, the most obvious interrogation is "what should be the next action?". The motor measurements must be included in s^i_β. Finally, the machine must decide what to do each time the query $C(s^*_\alpha, s^c_\beta)I(s^i_\beta)$ is answered. Except for global evaluations on scenes and lists of scenes, the MLM does not keep statistics on actions taken and does not calculate utilities. It uses instead a list of sensory-motor heuristics $[h_1, h_2, h_3 ... h_{\mathrm{rs}}]$, in the spirit of [8], to target or avoid the measurement results recorded in specific frame channels of s^c_β. Each heuristic defines the type of desired rewards resulting from the actions performed, and the type of past information and future continuations that are searched for a match in the DLM scenes. What is targeted is called *pleasure*, and what is avoided is called *pain*, and the corresponding frame channels are called *the pain-pleasure channels*. In this setting, anything can be pleasure or pain. But it makes sense, for instance, to assign nociceptors to the pain channels. The M-Logic Machine works with meaningless literal data in the frames, and the notions of pleasure and pain are operational. Roughly speaking, the MLM targets or avoids specific measurements (rewards) instead of evaluating utilities.

This leads to sub-rational choices, meaning sub-optimal rewards will be considered satisfactory as long as they are tagged pleasant. Pleasant tags are the tags that the sensory-motor heuristics try to make reappear in the future. Many frugal and simple heuristics - like "keep the same action as long as the latest resulting reward is positive" - are sub-rational in the sense that they do not look for any reward optimization. But they can make lots of sense in an evolutionary setting with cooperating and defecting agents [6][5]. In those scenarios survival does not require an optimization, just doing better than the competition. In the current MLM implementations, heuristics are placed in a list that is subject to an updating dominance mechanism that implements meta-learning. For a given situation, the most accurate heuristics tend to be used to select the most accurate continuations. New heuristics can be generated by new hardwired abilities or result from in-life exploratory learning.

3 COGNITIVE DRIVEN COOPERATION

3.1 The Paradox of Cooperation

Cooperation is obvious at all levels of the biosphere. Every living structure displays very complex inter-dependence and cooperation among its sub-structures. But the selfish competition paradigm is dominant in Evolution Theory, and cooperation does not arise as a stable strategy. This paradox of cooperation can be formulated inside Game Theory. The simplest case is when just two players P_1 and P_2 are considered and only two strategies are available to the players: cooperation C and defection D. Consider a payoff matrix for the first player and that this payoff matrix is identical for the second player, as shown in Table 3.

Table 3. Prisoner's Dilemma Payoff Table

	C	D
C	a	b
D	c	d

The most difficult setting for the establishment of cooperation is given by the prisoner's dilemma, where the inequalities $c > a > d > b$ hold. The difficulties of cooperation can also be found in other games, but they represent somewhat relaxed situations. To make this more easily perceived, let us assign specific values to the payoff table, as shown in Table 4.

Table 4. Prisoner's Dilemma Payoff Table

	C	D
C	1	-1
D	2	0

We notice that, if a first player changes from strategy D to strategy C while the second player keeps a D strategy, he will get a lesser reward. Since the second player uses exactly the same payoff table, he reaches the same conclusion: while the first player chooses D, it's not worth changing from D to C. But, whatever the opponent's choice, it is always advantageous to change unilaterally from C to D. The players will therefore reach an equilibrium point when they both defect. A combination of the player strategies that cannot be improved by a unilateral change of strategy by one of the players is called a Nash Equilibrium. Notice that choosing a Nash equilibrium appeals to the cognitive abilities of the players. It is assumed that players are aware of the structure of the game, are consciously attempting to maximize their payoffs, and are attempting to predict the moves of their opponents. In addition, it is presumed that all payoffs and all other cognitive aspects are known by all the players. These facts are then used to explain why players will choose Nash equilibrium strategies. Suppose this same game is iterated many times, with the players choosing simultaneously their next move without prior knowledge of each other's choice. Some fixed strategies that take into account the past choices of the other player can be envisaged that will bring a better average payoff than the Nash Equilibrium. A well known successful strategy is called Tit-for-Tat (TFT): it starts cooperating, and then it simply replicates the other player's previous choice. If both players use a TFT strategy, they will stick to cooperation. A somewhat different setting is assumed in Evolutionary Game Theory. Here, it is presumed that the players are individuals with biologically encoded, heritable strategies. The individuals have no control over the strategy they play and they are not supposed to change it driven by some internal cognitive mechanism. They need not even be aware they are involved in a survival game. During the game the individuals (associated with fixed strategies) reproduce and are subject to the forces of natural selection. The payoffs of the game represent biological fitness that impacts on the reproductive ability of the individuals. Therefore the proportion (or frequency) of each type of individuals (and strategies) will change through time. The changing frequencies will change the total payoff each individual receives while interacting with all the other players. Therefore the payoff tables are no longer static, but evolve according to the frequencies. The evolution of frequencies follows some given Replicator Dynamics. Alternative strategies can be occasionally inserted in the game, via a process similar to mutation. In order to be an Evolutionary Stable Strategy, a strategy must be resistant to invasions occasionally made by a few individuals that bring in mutant strategies. The replicator dynamics are often implemented assuming a constant number of individuals. At each step, one individual is randomly chosen to die while another is created, with its associated strategy chosen with a probability proportional to fitness. Since each individual is associated with a single strategy, this may also be seen as the random change of strategy of a single individual. It is known that any Evolutionary Stable Strategy corresponds to a Nash Equilibrium (with pure or mixed strategies), but there are some Nash Equilibria that are not Evolutionary Stable Strategies. Evolutionary Stable Strategies do not lead to cooperation in a prisoner's dilemma setting, and yet cooperation is ubiquitous in the living world, hence the paradox of cooperation. Some solutions have been proposed to solve this problem [6][5]. The basic idea is to add some structure to the interactions among individuals and populations, and allow more complex strategies that go beyond simple reflex actions. This requires individuals endowed with higher cognitive abilities. For instance, indirect reciprocity relies on the good or bad reputation of each individual, implying some form of social evaluation and symbolic communication.

3.2 Cognitive-Driven Cooperation

The M-Logic Machine does not quite fit in any of the above settings. As we saw in section 2.2, it does not use the fixed payoff tables needed for classical game theory. Since it does not know the tables, it cannot calculate beforehand the Nash equilibria. On the other side, it is not bound to a fixed strategy, or even a random strategy change according to fitness, as assumed in Evolutionary Game Theory. The M-Logic Machine uses instead a set of simple heuristics that become more or less dominant according to their predictive success. In a two-player setting, with the MLMs opponent following a Tit-For-Tat strategy, the machine will most often tend to cooperate, as long as the measurement for both players defecting is tagged as painful and that for both players cooperating is tagged as pleasant. Noting pain with pn and pleasure with pl, the literal data table for the prisoner's is shown in Table 5.

Table 5. Pain-Pleasure Table

	C	D
C	pl	pn
D	pl	pn

The MLMs heuristics will try to promote the occurrence of the pl tag in the scenes while trying to avoid the occurrence of pn. This is why pl is called pleasure and pn is called pain. Except for this relation to specific types of heuristics, pl and pn are meaningless tags. To strictly respect the $c > a > d > b$ inequalities we could have used specific heuristics for lots of pleasure and lots of pain and assign $pl+$ and $pn+$ tags to these heuristics. But this would not change the dynamics of the heuristics in the MLM, since the machine does not give any a priori preference to the heuristics. It is easily seen that, playing against a Tit-For-Tat strategy, the choice C (cooperate) will most often succeed when the MLM uses heuristics that search for pleasure. If the opponent chooses a Always Defect strategy the MLM will soon find that no heuristic can bring pleasure, and will oscillate randomly between C (cooperate) and D (defect). Although this is clearly a sub-rational choice, it makes lots of sense in a world that is not assumed stationary. Facing constant pain, the MLM just keeps exploring. This particular configuration of the literal data table came from the chosen split value between pleasure and pain compatible with the $c > a > d > b$ inequalities. But other compatible split points can also be considered. Suppose the MLM considers pleasant the result from both players defecting, as shown in Table 6.

Table 6. Pain-Pleasure Table

	C	D
C	pl	pn
D	pl	pl

Now the MLMs search for pleasure will be most often satisfied with choice D (defect). The same result is reached when the MLM only finds pleasure in cheating its opponent, as shown in Table 7.

Table 7. Pain-Pleasure Table

	C	D
C	pn	pn
D	pl	pn

On the other extreme, if the MLM takes pleasure even in the success of the defector, we get the configuration shown in Table 8. In this case, the MLM will tend to reuse any initial random sequence of choices. This is called *superstitious learning* [8].

Table 8. Pain-Pleasure Table

	C	D
C	pl	pl
D	pl	pl

4 FINAL REMARKS

We thus see that reaching stable cooperation with an M-Logic Machine in a prisoner's dilemma setting strongly depends on the split value between pleasure and pain. Only when cooperation is perceived as good and defection is perceived as bad does the MLM tend to cooperate. In any case, the MLM only sticks to cooperation as long as the opponent also tends to cooperate. This result agrees with our intuitive justification for a cognitive-driven cooperation in the prisoner's dilemma. The M-Logic Machine's cognitive mechanism therefore offers an interesting alternative to the Evolutionary Game Theory approach.

The M-Logic Machine is somewhat peculiar, in the sense that it pushes to the limit the avoidance of statistical calculations. Working mainly with ”literal data” (tags identifying the possible outputs of measurement instruments), it cannot calculate averages or standard deviations. Only the mode can be captured by the update dominance processes. The MLM does not calculate state utilities using rewards, discount rates, learning factors, etc. The idea of rewards emerges from the machine's processes. The MLM efficiency relies on the integration of processes and features that are seldom considered in mainstream AI: orienting reflexes, sensory modes, frugal heuristics, clever measurements, etc. Much remains to be done in these areas, but the current MLM implementation works well enough to show the approach is worth considering. Although some similarities can be found with Temporal Difference Learning, Q-Learning, and Instance-Based Learning (all explained in [7]), the MLM approach only makes sense inside a unified theory of cognition.

REFERENCES

[1] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, ‘An integrated theory of the mind.’, *Psychol Rev*, **111**(4), 1036–1060, (October 2004).

[2] J.F. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to soar (2006 update). http://ai.eecs.umich.edu/soar/.

[3] G. A. Miller, ‘The magical number seven plus or minus two: some limits on our capacity for processing information.’, *Psychol Rev*, **63**(2), 81–97, (March 1956).

[4] Allen Newell, *Unified theories of cognition*, Harvard University Press, Cambridge, MA, USA, 1990.

[5] M.A. Nowak, C.E. Tranita, and T. Antal, ‘Evolutionary dynamics in structured populations’, *Phil. Trans. R. Soc. B*, **365**, 19–30, (2010).

[6] Martin A. Nowak, ‘Five rules for the evolution of cooperation’, *Science*, **314**(5805), 1560–1563, (December 2006).

[7] S. Russel and P. Norvig, *Artificial Intelligence: a modern approach*, Prentice Hall Series in Artificial Intelligence, New Jersey, 1995.

[8] B. F. Skinner, ‘’superstition’ in the pigeon.’, *J Exp Psychol Gen*, **121**(3), 273–274, (September 1992).

[9] E. Sokolov, ‘The orienting response, and future directions of its development’, *Integrative Psychological and Behavioral Science*, **25**(3), 142–150, (July 1990).

[10] N.E. Sokolov, H. Lyytinen, R. Naatanen, and J.A. Spinks, *The Orienting Response in Information Processing*, Lawrence Erlbaum Associates, New Jersey, 2002.

[11] R. Sun, ‘Clarion 5.0 technical report’, Technical report, Cognitive Science Department, Rensselaer Polytechnic Institute, (2003).

[12] *The Cambridge Handbook of Computational Psychology*, ed., R. Sun, Cambridge University Press, 1st edn., April 2008.

[13] Ron Sun, ‘Theoretical status of computational cognitive modeling’, *Cognitive Systems Research*, **10**(2), 124–140, (June 2009).

3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)
Lisboa, Portugal, August 16 2010

A Heuristic Strategy for Learning in Partially Observable and Non-Markovian Domains

Matteo Leonetti [1] and **Subramanian Ramamoorthy** [2]

Abstract. Robotic applications are characterized by highly dynamic domains, where the agent has neither full control of the environment nor full observability. In those cases a Markovian model of the domain, able to capture all the aspects that the agent might need to predict, is generally not available or excessively complex. Moreover, robots pose relevant constraints on the amount of experience they can afford, moving the focus of learning their behavior from reaching optimality in the limit, to making the best use of the little information available. We consider the problem of finding the best deterministic policy in a Non-Markovian Decision Process, with a special attention to the sample complexity and the transitional behavior before such a policy is reached. We would like robotic agents to learn in real time while being deployed in the environment, and their behavior to be *acceptable* even while learning.

1 Introduction

Robotic applications are characterized by highly dynamic domains, where the agent has neither full control of the environment nor full observability. In those cases a Markovian model of the domain, able to capture all the aspects that the agent might need to predict, is generally not available or excessively complex. A very general framework to face such problems is the one of Partially Observable MDP (POMDP) [2].

While most of the methods to solve POMDPs attempt some state estimation, we follow the previous work in the literature about learning with hidden states [6, 5, 4] (which make the system Non-Markovian in general), and focus on a different aspect of the learning process. In Reinforcement Learning (RL) optimality is usually the main target, but robotic applications pose relevant constraints on the number of experiments that the agent can afford (in terms of time, or other resources). We, therefore, believe that the focus should be moved from proving optimality in the limit, to obtaining the best possible behavior with the little information available, and gathering this information carefully.

We consider the problem of finding the best deterministic policy in a Non-Markovian Decision Process, with a special attention to the sample complexity and the transitional behavior before such a policy is reached. We would like robotic agents to learn in real time while being deployed in the environment, and their behavior to be *acceptable* even while learning. To this aim, we propose an algorithm structured in two phases: the first one, as short as possible unless simulated, provides an exploratory behavior that gathers information on the effect of actions.The second phase starts exploiting the data collected during the first phase making small exploratory steps and traversing the policies that look more promising on the basis of the collected data.

As an example, consider the domain of robotic soccer, in which multiple agents interact in both a cooperative and a competitive way, making the environment extremely dynamic and unpredictable from a single-agent perspective. Moreover, in applications such as the one just mentioned, the number of actions available at any time is considerable, making the branching factor of the policies an issue. Nonetheless, the actions actually meaningful in most of the situations are few. A robot should ideally be able to realize quickly that, for instance, just staring at the ball is not going to take it in any farther, no matter what other actions he could do later, and independently from all the other many aspects of the world. The method we propose aims at identifying those "wrong" actions and avoiding them unless proved necessary.

We provide a preliminary evaluation on a common test-bed in the literature of NMDP that allows us to easily compare our algorithm with the best results obtained so far.

2 Problem formulation

We consider NMDPs with a finite set of states S and a finite set of actions A, with similar assumptions regarding observations as in POMDPs. A deterministic policy π on the NMDP maps each state $s \in S$ to an action $a \in A(s)$ among those available in s. In the following, we borrow the notation from Perkins [5] indicating with $\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T\}$ a trajectory in the NMDP. A policy determines a well defined probability measure, $\mu(\pi)$, over the set of all possible trajectories. The reward corresponding to each trajectory is a random variable defined as $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ where $\gamma \in [0, 1]$ is the discount factor. We define the value of a policy as the expected discounted reward:

$$V^\pi = E_{\tau \sim \mu(\pi)}\{R(\tau)\} = E^\pi\{R(\tau)\}$$

The tasks are episodic, which means that they terminate under any policy with probability one. We limit ourselves to the search for the best deterministic policy, although in NMDPs the optimal policy might be stochastic [6]. We also adopt Perkins's [5] definition of the action-value function that we report in the following. Given a trajectory τ, the portion of $R(\tau)$ preceding a state s is denoted as $R_{pre-s}(\tau)$. Similarly, the portion of $R(\tau)$ following a state s is denoted as $R_{post-s}(\tau)$. For any state s the value of a policy can be rewritten as

$$\begin{aligned} V^\pi &= E^\pi[R(\tau)] \\ &= E^\pi[R_{pre-s}(\tau)] + E^\pi[R_{post-s}(\tau)] \end{aligned}$$

[1] Department of Computer and System Sciences, Sapienza University of Rome
[2] School of Informatics, The University of Edinburgh

Let $\pi \leftarrow (s,a)$ represent the policy that is identical to π except for the state s that is mapped to the action a. We define the action-value function for a pair $\langle s,a \rangle$ as:

$$Q^{\pi}(s,a) = E^{\pi \leftarrow (s,a)}[R_{post-s}(\tau)] \tag{1}$$

We refer to the original paper for an explanation of the differences with the traditional definition. We only point out that if an action a is chosen for a state s, then every time s is encountered the agent will execute a, according to $\pi \leftarrow (s,a)$.

In the following, we present an algorithm for maximizing $Q^{\pi}_{s,a}$ making use of a particular initial exploration phase to collect heuristic information on the most promising policies to be subsequently exploited.

3 Parr and Russell's Grid World

Before proceeding with the description of the algorithm, we introduce the test domain: Parr and Russell's Grid World [3]. Grid World has been used as a test domain in several papers [3, 1, 5] and provides a simple and structured environment with a reasonable branching factor. It has 11 states (figure 1) in a 4 by 3 grid with one obstacle. The agent starts at the bottom left corner. There is a target state and

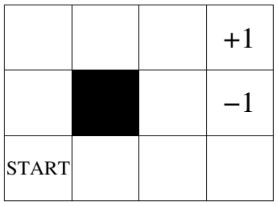


Figure 1. Grid World

a penalty state whose rewards are +1 and -1 respectively. Both are absorbing states, that is when the agent enters them the episode terminates. The actions available in every state are *move north*, *move south*, *move east*, and *move west* which succeed with probability 0.8. With probability 0.1 the agent moves in one of the directions orthogonal to the desired one. In all of the previous cases if the movement is prevented by an obstacle the agent stays put. In any state the agent can only observe the squares east and west of it, having a total of four possible observations. Those observations are going to form the state space of an NMDP whose controller we are going to learn.

4 The algorithm: ϵMaCs

The main idea of the algorithm lies on the intuition that often a few bad choices disrupt the value of all the policies that include them. For instance, consider the initial state in Grid World. Any of the 128 policies out of the 256 total ones that map the initial observation to either *move west* or *move south* have no chance to be optimal. Taking those policies as if they were as valuable as any other, in the search for the optimal policy, just wastes samples. We would rather like to realize that those actions are not promising and not consider them unless we have tried all the other possibilities.

The strategy would ideally consider all the policies from the most *promising* to the least ones, which we believe is beneficial in at least two ways: (1) the algorithm reaches the optimal policy earlier; (2) during the phase of evaluation of those *promising* but suboptimal policies, the behavior is as good as the current information allows.

The algorithm is constituted by two parts: the *exploratory* phase and the *assessing* phase.

Algorithm 1 ϵMaCs

```
$exp\_length \leftarrow$ number of episodes in the exploratory phase
$\epsilon \leftarrow$ probability of exploration in the assessing phase
$\alpha \leftarrow$ learning step parameter
initialize $Q(s,a)$ pessimistically
{Exploratory phase}
for $i = 1$ to $exp\_length$ do
    generate a trajectory $\tau$ according to a policy $\pi$ extracted uniformly at random
    for all $s \in S, a \in A\ s.t. \langle s,a \rangle$ is in $\tau$ do
        $Q(s,a) = max(Q(s,a), R_{post-s}(\tau))$
    end for
end for
{Assessing phase}
for all other episodes do
    $v \leftarrow$ a value in $[0,1]$ uniformly at random
    if $v \geq \epsilon$ then
        $\pi' \leftarrow$ the policy that greedily maximizes Q
    else
        $\pi' \leftarrow$ a policy chosen uniformly at random
    end if
    generate a trajectory $\tau$ from $\pi'$
    if $v \geq \epsilon$ then
        for all $s \in S, a \in A\ s.t. \langle s,a \rangle$ is in $\tau$ do
            $Q(s,a) = (1-\alpha)Q(s,a) + \alpha R_{post-s}(\tau)$
        end for
    else
        for all $s \in S, a \in A\ s.t. \langle s,a \rangle$ is in $\tau$ do
            $q = (1-\alpha)Q(s,a) + \alpha R_{post-s}(\tau)$
            $Q(s,a) = max(Q(s,a), q)$
        end for
    end if
end for
```

4.1 Exploration: gathering information

The exploration initializes the Q-function to drive the execution in the subsequent phase. For a number of episodes exp_length the agent chooses a policy at random, and in each pair $\langle s,a \rangle$ stores the highest value that any policy, going through $\langle s,a \rangle$, has obtained until then. Consider the simple example of the NMDP in figure 2(a). This NMDP has three states and four actions with a total of four policies. Let the reward returned by each of those policies be normally distributed, with means and standard deviations represented in figure 2(b). Figure 2(c) and 2(d) show the value of the Q-function for each action during a particular run. The first 100 episodes belong to the exploratory phase, in which A1 and A2 obtain the highest reward, making the policy A1-A2 look particularly promising. An action is considered as *promising* as the highest value of the reward that choosing that action has ever given. In the case of A1-A2, its good result is due to the high variance, rather than the highest mean. This aspect is going to be addressed by the second phase of the algorithm.

Several other choices are possible both for the exploration (uniformly at random) and for the value stored (the maximum); for instance, making use of SoftMax we might give a higher priority to the

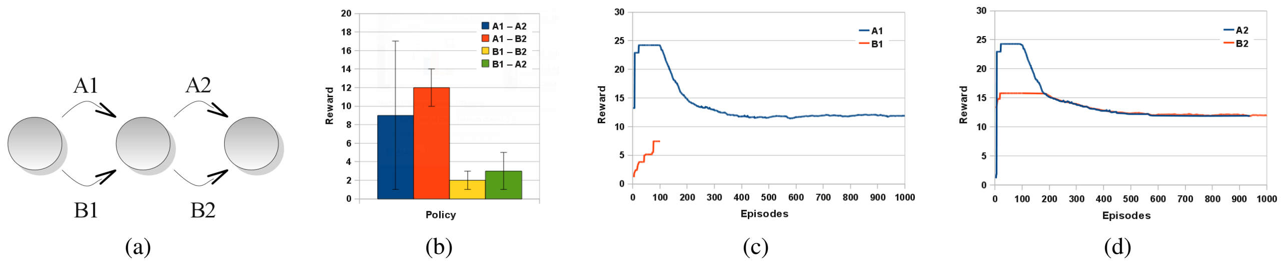


Figure 2. A simple example of an NMDP (a). The four policies return a reward normally distributed whose means and standard deviations are shown in (b). The evolution of the Q-function for the first state (actions A1 and B1) is represented in figure (c), while for the second state (actions A2 and B2) is represented in figure (d).

policies that returned a higher reward. In future work, we will consider relying on further statistics about the policies traversing a given choice point.

4.2 Assessment

We want to maximize the *expected* cumulative discounted reward, rather than the maximum obtainable one, therefore an evaluation of the *promising* policies is needed.

In the second phase the agent acts greedily according to the Q-function previously constructed. It starts from the policy that has given the highest reward and, taking subsequent samples from it, lowers its value until it reaches the expected value. The value of the current policy, in this process, could get lower than the maximum value obtained by some other policy, and stored in one of its actions. In this case, the two policies would be alternately chosen and would race each other down toward their respective expected values, stopping at the highest one. An example of this behavior is shown in figure 2(d). Starting from the episode 101, the policy A1-A2 is greedily executed. Since its mean is considerably lower than the value stored, the estimate keeps dropping until it reaches the value previously stored for B2. At this point, A1-A2 and A1-B2 are executed almost alternately, each one pushing down its estimate toward their respective mean. The learning rate parameter determines the speed with witch the estimates tend to the means. The higher the learning rate, the faster the estimate will reach the mean, but also, the more it oscillates confusing policies close to each other. At some point, A1-B2 reaches its mean and stabilizes, so that the value of A1 is the same for both the policies, but B2 is definitely higher than A2, making A1-B2 the learned policy which is also the optimal one. Notice how in this process B1 has never been executed. This is because the highest value it had given in the first phase has always been lower than the current estimates of the policies taken into account during the assessing phase.

If the maximum value obtained by the optimal policy has been stored in one of the state-action pairs, this algorithm quickly converges to the optimal policy. Unfortunately, this might not happen for two reasons: (1) the optimal policy might have, in all of its states, another policy (not the same for every state) that shares the same action in that state and differs elsewhere, but gives a higher maximum reward. In this case the maximum reward obtained by the optimal policy in the first phase would be hidden by those policies. (2) The optimal policy has never been sampled above the expected reward, and no optimistic estimate of it has had a chance to be stored. Ideally, in the first phase, every policy should be sampled above its expected value at least once. The number of episodes necessary to meet this condition would probably be impractically high for most domains. For this reason, and since the first point wouldn't be avoidable anyway, we add a step of exploration in the second phase too, that guarantees that each policy is continually sampled on the long term. When the agent takes an exploratory step (with probability ϵ) the Q-function is updated only in those state-action pairs that gave a value higher than the current one.

Clearly taking an exploratory step could disrupt the optimal policy if this had already been found. The racing among the two policies would have to happen again until the optimal policy is established once more. This cannot be prevented if we want to make sure that the optimal policy has always a chance to be sampled.

5 Experimental evaluation

We conducted a preliminary evaluation of our algorithm on Grid World, in order to show how the different parameters impact the behavior of the agent. Every 20 episodes for the short term experiments, and every 100 episodes for the long term ones, we pause the learning and evaluate the current controller for 20 episodes. The results are averaged over 200 runs. By "evaluating the controller" we mean that, during the evaluation, the behavior of the agent is the same as if it were learning, but the Q-function is left unchanged. Thus, if at a specific point the agent would choose a policy at random, the reward obtained will be the average of the reward returned following 20 policies picked uniformly at random. Notice that choosing a policy at random, in this context, is different from following the *random policy*. In the former case the same decision is always made in the same state, while in the latter case each time a state is hit a random choice is made.

We compare our results with two control strategies: Sarsa(λ) with ϵ-greedy exploration, and Sarsa(λ) with optimistic initialization. The latter strategy consists in initializing the Q-function at an optimistic value for each state-action pair, and exploiting the current estimate at any time. We borrowed a few parameters from the literature [1, 5] and spent some time optimizing others. When not differently specified the Q-function has been initialized at -4. The best behavior we could achieve for ϵ-greedy was with ϵ starting at 0.2 and linearly decaying to 0 in 80000 actions. For the optimistic initialization, the Q-function has been initialized at 1. In both cases $\alpha = 0.01$ and $\lambda = 0.9$

Figure 3 shows the cumulative rewards obtained by different controllers. ϵMaCs here has been evaluated without any exploration in its second phase. Sarsa(0.9) with optimistic initialization reliably converges to the optimal policy a lot faster than ϵ-greedy. It is this behavior that we want to improve, pruning some of the exploration by getting a more realistic initialization. With an initial phase of 100

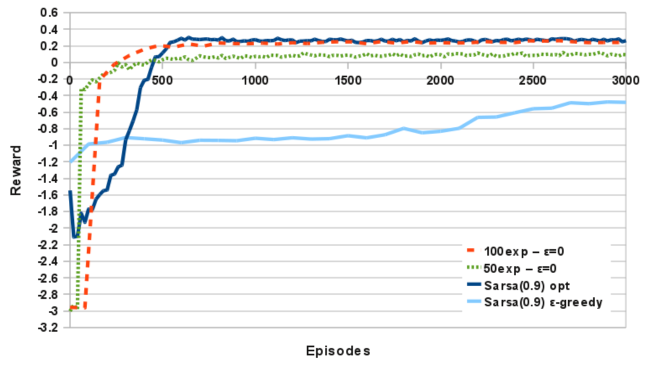


Figure 3. Cumulative reward on the short term for different controllers. ϵMaCs is evaluated without any exploration in the second phase.

episodes and $\alpha = 0.01$, ϵMaCs always converges to the optimal policy shortly after the initial exploration. We also evaluated the behavior of the agent with 50 episodes, in order to understand the consequences of little initial sampling. In case the agent could not afford a longer initial phase, we would like that it still quickly converged to a "good" policy, if not the optimal one. Indeed, after 50 episodes the average reward stabilizes at around 0.1, while the optimum is around 0.25. Considering that most of the policies give a reward of -4 and that the average reward obtained is non-decreasing, this can be probably considered a good result.

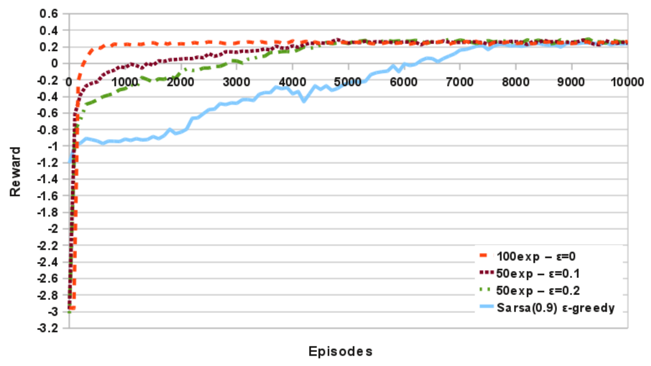


Figure 4. Cumulative reward on the long term allowing exploration in the second phase of ϵMaCs

In the second set of experiments we tried to establish whether, by allowing some exploration also on the second phase, it is eventually possible to reach the optimal policy even from a short initial phase. Clearly exploration is a double-edged sword: on the one hand it allows to discover the optimal policy, on the other hand it worsen the average behavior of the agent that leaves its current "good" policy. Figure 4 shows the results for two different settings, compared with Sarsa(λ) and ϵMaCs after 100 initial episodes as already described. In one setting we let ϵ start at 0.2 and reach 0.01 in 5000 episodes, remaining constant afterwords. In the other setting ϵ started at 0.1. The two results fall in between Sarsa and the optimum obtained with 100 initial episodes. Moreover, the increase in the performance is linear and follows perfectly the decay of ϵ. This probably means that the optimal policy is identified early and, from that point on, the exploration is the only responsible for the sub-optimal behavior. We have not performed an extensive evaluation over the possible values for the initial ϵ and its decaying rate, therefore we cannot state exactly how close the behavior can be pushed towards the optimal line above by varying these two values. It seems reasonable though, that the linear dependence allows for a faster convergence up to a point when the exploration becomes too short, and we fall into the initial case of figure 3 with no exploration at all.

6 Conclusion

We devised an algorithm to learn the best deterministic policy in an NMDP searching the policy space in a favorable order. The algorithm first attempts to collect information about the actions' values and then exploits it preventing the agent from behaving arbitrarily bad, possibly allowing its early deployment in the environment. Several future directions can be followed in making this simple algorithm more efficient in the exploration, directing the search even in the first phase rather than acting randomly. We believe that novel and more efficient techniques on NMDPs can help to the simplifications of robots' controllers and the scalability of RL in practical applications.

REFERENCES

[1] J. Loch and S. Singh, 'Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes', in *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 323–331. Citeseer, (1998).

[2] G.E. Monahan, 'A survey of partially observable Markov decision processes: Theory, models, and algorithms', *Management Science*, **28**(1), 1–16, (1982).

[3] R. Parr and S. Russell, 'Approximating optimal policies for partially observable stochastic domains', in *Ineternational Joint Conference on Artificial Intelligence*, volume 14, pp. 1088–1095. Citeseer, (1995).

[4] Mark D. Pendrith and Michael McGarity, 'An analysis of direct reinforcement learning in non-markovian domains.', in *ICML*, ed., Jude W. Shavlik, pp. 421–429. Morgan Kaufmann, (1998).

[5] T.J. Perkins, 'Reinforcement learning for POMDPs based on action values and stochastic optimization', in *Proceeding of the national conference on artificial intelligence*, pp. 199–204. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, (2002).

[6] S.P. Singh, T. Jaakkola, and M.I. Jordan, 'Learning without state-estimation in partially observable Markovian decision processes', in *Proceedings of the eleventh international conference on machine learning*, pp. 284–292. Citeseer, (1994).