The 18th European Conference on Artificial Intelligence

Proceedings of the

1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2008)

Tuesday, July 22 2008
Patras, Greece

Nils T Siebel and Josef Pauli

http://www.erlars.org/

Table of Contents

A Message from the Chairs

We would like to welcome you all to the 1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems, ERLARS 2008, held in conjunction with the ECAI 2008 conference in Patras, Greece on July 22 2008.

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 that you will find 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.

We would like to thank the program committee members who provided excellent reviews in a short period of time. We are also especially indebted to the authors of the articles that were sent to this workshop because they provided 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 and Josef Pauli, Chairs, ERLARS 2008 Workshop.

Organisation of the ERLARS 2008 Workshop

Workshop Chairs

Nils T Siebel
Cognitive Systems Group
Institute of Computer Science
Christian-Albrechts-University of Kiel
Kiel, Germany

Josef Pauli
Intelligent Systems Group
Department of Computer Science
University of Duisburg-Essen
Duisburg, 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.
Yohannes Kassahun, DFKI Lab Bremen, University of Bremen, Germany.
Takanori Koga, Kyushu Institute of Technology, Kitakyushu, Japan.
Tim Kovacs, Department of Computer Science, University of Bristol, UK.
Jun Ota, Graduate School of Engineering, University of Tokyo, Japan.
Josef Pauli, Intelligent Systems Group, University of Duisburg-Essen, Germany.
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.
Nils T Siebel, Institute of Computer Science, Christian-Albrechts-University of Kiel, Germany.
Marc Toussaint, Berlin Machine Learning and Robotics Group, TU Berlin, Germany.
Jeremy Wyatt, School of Computer Science, University of Birmingham.

1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2008)
Patras, Greece, July 22 2008

Gradient Based Reinforcement Learning for Autonomous Underwater Cable Tracking

Andres El-Fakdi and **Marc Carreras** and **Emili Hernandez**[1]

Abstract. This paper proposes a field application of a high-level Reinforcement Learning (RL) control system for solving the action selection problem of an autonomous robot in a cable tracking task. The learning system is characterized by using a policy gradient based search method for learning the internal state/action mapping. The function approximator used to represent the policy is a barycentric interpolator. Policy only algorithms may suffer from long convergence times when dealing with real robotics. In order to speed up the process, the learning phase has been carried out in a simulated environment using the hydrodynamic model of the vehicle and, in a second step, the policy has been transferred and tested successfully on a real robot. Future steps plan to continue the learning process on-line while on the real robot while performing the mentioned task. We demonstrate its feasibility with real experiments on the underwater robot ICTINEU Autonomous Underwater Vehicle (AUV).

1 INTRODUCTION

Reinforcement Learning (RL) is a widely used methodology in robot learning, see [23]. In RL, an agent tries to maximize a scalar evaluation obtained as a result of its interaction with the environment. The goal of a RL system is to find an optimal policy to map the state of the environment to an action which in turn will maximize the accumulated future rewards. The agent interacts with a new, undiscovered environment selecting actions for each state, receiving a numerical reward for every decision. Obtained rewards are used to teach the agent so the robot learns which action to take at each state, achieving an optimal or sub-optimal policy (state-action mapping).

The dominant approach over the last decade has been to apply reinforcement learning using the value function approach. Although value function methodologies have worked well in many applications, they have several limitations. The considerable amount of computational requirements that increase time consumption and the lack of generalization among continuous variables represent the two main disadvantages of "value" RL algorithms. Over the past few years, studies have shown that approximating a policy can be easier than working with value functions, and better results can be obtained ([24] [2]). As presented in [1], it is intuitively simpler to determine *how to act* instead of *value of acting*. So, rather than approximating a value function, new methodologies approximate a policy using an independent function approximator with its own parameters, trying to maximize the future expected reward. Only a few but promising practical applications of policy gradient algorithms have appeared, this paper emphasizes the work presented in [5], where an autonomous helicopter learns to fly using an off-line model-based policy search method. Also important is the work presented in [21] where a simple "biologically motivated" policy gradient method is used to teach a robot in a weightlifting task. More recent is the work done in [10] where a simplified policy gradient algorithm is implemented to optimize the gait of Sony's AIBO quadrupedal robot. More recently, the work presented in [18] gives an overview on learning with policy gradient methods for robotics while presenting the results obtained in the application of hitting a baseball with an anthropomorphic arm.

All these recent applications share a common drawback, gradient estimators used in these algorithms may have a large variance (see [13] and [11]) what means that policy gradient methods learn much more slower than RL algorithms using a value function (see [24]) and they can converge to local optima of the expected reward (see [15]), making them less suitable for on-line learning in real applications. In order to decrease convergence times and avoid local optima, newest applications combine policy gradient algorithms with other methodologies, it is worth to mention the work done in [25] and [14], where a biped robot is trained to walk by means of a "hybrid" RL algorithm that combines policy search with value function methods.

A good proposal for speeding up gradient methods may be offering the agent an initial policy. Example policies can direct the learner to explore the promising part of search space which contains the goal states, specially important when dealing with large state-spaces whose exploration may be infeasible. Also, local maxima dead ends can be avoided with example techniques [12]. The idea of providing high-level information and then use machine learning to improve the policy has been successfully used in [22] where a mobile robot learns to perform a corridor following task with the supply of example trajectories. In [4] the agent learns a reward function from demonstration and a task model by attempting to perform the task. Finally, the work done in [9] concerning an outdoor mobile robot that learns to avoid collisions by observing a human driver operating the vehicle.

This paper proposes a reinforcement learning application where the underwater vehicle $ICTINEU^{AUV}$ carries out a visual based cable tracking task using a direct gradient algorithm to represent the policy. The function approximator used to represent the policy a barycentric interpolator function. An initial example policy is first computed by means of computer simulation where a hydrodynamic model of the vehicle simulates the cable following task. Once the simulated results are accurate enough, in a second phase, the policy is transferred to the vehicle and executed in a real test. A third step will be mentioned as a future work, where the learning procedure continues on-line while the robot performs the task, with the objective of improving the initial example policy as a result of the interaction with the real environment. This paper is structured as follows. In Section 2 the learning procedure and the policy gradient algorithm are detailed. Section 3 describes all the elements that affect our problem: the underwater robot $ICTINEU^{AUV}$, the mathematical model of

[1] University of Girona, Spain, email: aelfakdi@eia.udg.edu

the vehicle used in the simulation, the vision system and the controller. Details and results of the simulation process and the real test are given in Section 4 and finally, conclusions and the future work to be done are included in Section 5.

2 LEARNING PROCEDURES

The introduction of prior knowledge in a gradient descent methodology can dramatically decrease the convergence time of the algorithm. This advantage is even more important when dealing with real systems, where timing is a key factor. Such learning systems can divide its procedure into two phases or steps as shown in Fig. 1. In the first phase of learning (see Fig. 1(a)) the learner interacts with a simulated environment; during this phase, the agent extracts all useful information from simulation. In a second step, once it is considered that the agent has enough knowledge to build a "secure" policy, it takes control of the real robot and the learning process continues in the real world, see Fig. 1(b).

The proposal presented here takes advantage of learning by simulation as an initial startup for the learner. The objective is to transfer an initial policy, learned in a simulated environment, to a real robot and test the behavior of the learned policy in real conditions. First, the learning task will be performed in simulation with the aid of the hydrodynamic model of the robot. Once the learning process is considered to be finished, the policy will be transferred to $ICTINEU^{AUV}$ in order to test it in the real world. In a future task, the learning procedure will switch to a third phase, continuing to improve the policy while in real conditions. The Baxter and Bartlett approach [6] is the gradient descent method selected to carry out the simulated learning corresponding to phase one. Next subsection gives details about the algorithm.

2.1 The gradient descent algorithm

The Baxter and Bartlett's algorithm is a policy search methodology with the aim of obtaining a parameterized policy that converges to an optimum by computing approximations of the gradient of the averaged reward from a single path of a controlled *partially observable Markov decision process* (POMDP). The convergence of the method is proven with probability 1, and one of the most attractive features is that it can be implemented on-line. In a previous work presented in [7], the same algorithm was used in a simulation task achieving good results. The algorithm's procedure is summarized in Algorithm 1.

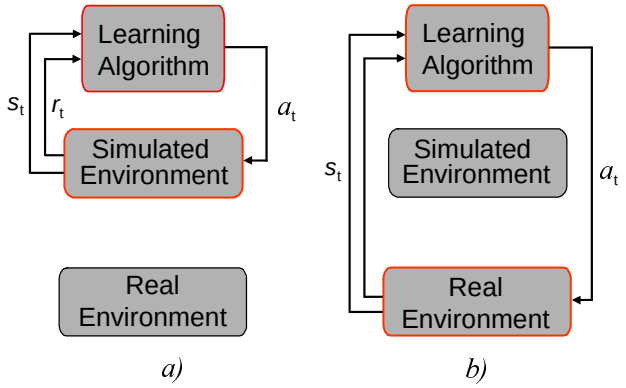


Figure 1. Learning phases.

The algorithm works as follows: having initialized the parameters vector θ_0, the initial state i_0 and the eligibility trace $z_0 = 0$, the learning procedure will be iterated T times. At every iteration, the parameters' eligibility z_t will be updated according to the policy gradient approximation. The discount factor $\beta \in [0, 1)$ increases or decreases the agent's memory of past actions. The immediate reward received $r(i_{t+1})$, and the learning rate α allows us to finally compute the new vector of parameters θ_{t+1}. The current policy is directly modified by the new parameters becoming a new policy to be followed by the next iteration, getting closer to a final policy that represents a correct solution of the problem.

Algorithm 1: Baxter and Bartlett's OLPOMDP algorithm

1. Initialize:
   $T > 0$
   Initial parameter values $\theta_0 \in R^K$
   Initial state i_0
2. Set $z_0 = 0$ ($z_0 \in R^K$)
3. $for\ t = 0$ to T do:
   (a) Observe state x_t
   (b) Generate control action u_t according to current policy $\mu(\theta, x_t)$
   (c) Observe the reward obtained $r(i_{t+1})$
   (d) Set $z_{t+1} = \beta z_t + \frac{\nabla \mu_{u_t}(\theta, x_t)}{\mu_{u_t}(\theta, x_t)}$
   (e) Set $\theta_{t+1} = \theta_t + \alpha_t r(i_{t+1}) z_{t+1}$
4. end

The algorithm is designed to work on-line. Our policy will be approximated with a particular class of functions called the *barycentric interpolators* (see [16]), which use an interpolation process based on a finite set of discrete points conforming a mesh. This mesh does not need to be regular, but the method outlined here assumes that the state space is divided into a set of rectangular boxes as shown in Fig. 2. The key here is that any particular point is enclosed in a rectangular box that can be defined by 2^N nodes in our N-dimensional state space.

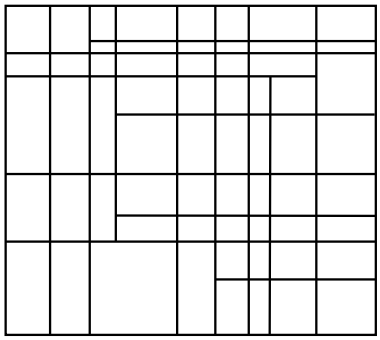


Figure 2. Example of a 2-dimension rectangular mesh.

Let $\Sigma^\delta = \{\xi_i\}_i$ be a set of points distributed in the mesh at some resolution δ on the state of dimension d. For any state x inside some rectangular box $(\xi_i, ..., \xi_n)$, x is the *barycenter* of the $\{\xi_i\}_{i=1..n}$ inside this box with positive coefficients $p(x|\xi_i)$ of sum 1 called the *barycentric coordinates* (see Fig. 3) where:

$$x = \sum_{i=1..n} p(x|\xi_i)\xi_i \qquad (1)$$

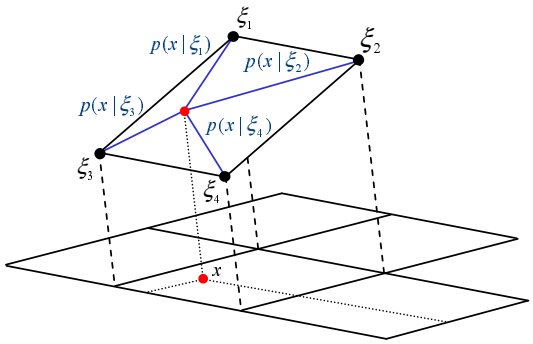


Figure 3. Graphic representation of the *barycentric coordinates* given a state x in a 2 dimensional mesh case.

We can set $V^{\delta}(\xi_i)$ as the value of the function at the points previously described ξ_i. As seen in Eq. 2, $V^{\delta}(x)$ is the **barycentric interpolator** of state x which is the barycenter of the points $\{\xi_i\}_{i=1..n}$ for some box $(\xi_i, ..., \xi_n)$ with barycentric coordinates $p(x|\xi_i)$, see Fig. 4.

$$V^{\delta}(x) = \sum_{i=1..n} p(x|\xi_i) V^{\delta}(\xi_i) \tag{2}$$

As stated before, our policy will be directly approximated using a barycentric interpolator function whose values $V^{\delta}(\xi_i)$ represent the policy parameters. Therefore, given and input state x_t the policy will compute a continuous control action $V^{\delta}(x)_t = u_t$ driving the learner to a new state with its associated reward. Once the action has been selected, the parameter update process starts. The barycentric interpolator parameters are updated following expression 3.(d) of Algorithm 1:

$$z(\xi_i)_{t+1} = \beta z(\xi_i)_t + \frac{\nabla \mu_{u_t}(V^{\delta}(\xi_i), x_t)}{\mu_{u_t}(V^{\delta}(\xi_i), x_t)} \tag{3}$$

$$z(\xi_i)_{t+1} = \beta z(\xi_i)_t + p(x|\xi_i) e \tag{4}$$

the error at the output is given by:

$$e = V^{\delta}(x)_{desired} - V^{\delta}(x) \tag{5}$$

$$z(\xi_i)_{t+1} = \beta z(\xi_i)_t + p(x|\xi_i)(V^{\delta}(x)_{desired} - V^{\delta}(x)) \tag{6}$$

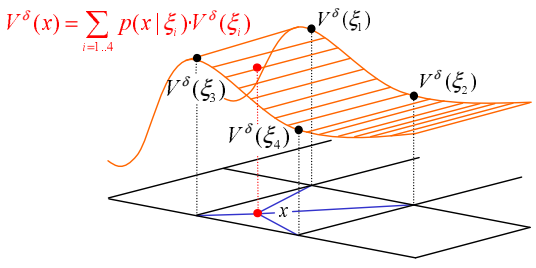


Figure 4. Calculation of the function approximator output given a particular state x.

Finally, the old parameters are updated following expression 3.(e) of Algorithm 1:

$$V^{\delta}(\xi_i)_{t+1} = V^{\delta}(\xi_i)_t + \alpha r(i_{t+1}) z_{t+1} \tag{7}$$

The vector $V^{\delta}(\xi_i)$ represents the policy parameters to be updated, $r(i_{t+1})$ is the reward given to the learner at every time step, z_{t+1}describes the estimated gradients mentioned before and, at last, we have α as the learning rate of the algorithm.

3 CASE TO STUDY: CABLE TRACKING

This section is going to describe the different elements that take place into our problem: first, a brief description of the underwater robot $ICTINEU^{AUV}$ and its model used in simulation is given. The section will also present the problem of underwater cable tracking and, finally, a description of the barycentric interpolator function designed for both, the simulation and the real phases is detailed.

3.1 $ICTINEU^{AUV}$

The underwater vehicle $ICTINEU^{AUV}$ was originally designed to compete in the SAUC-E competition that took place in London during the summer of 2006 [19]. Since then, the robot has been used as a research platform for different underwater inspection projects which include dams, harbors, shallow waters and cable/pipeline inspection.

The main design principle of $ICTINEU^{AUV}$ was to adopt a cheap structure simple to maintain and upgrade. For these reasons, the robot has been designed as an open frame vehicle. With a weight of 52 Kg, the robot has a complete sensor suite including an imaging sonar, a DVL, a compass, a pressure gauge, a temperature sensor, a DGPS unit and two cameras: a color one facing forward direction and a B/W camera with downward orientation. Hardware and batteries are enclosed into two cylindrical hulls designed to withstand pressures of 11 atmospheres. The weight is mainly located at the bottom of the vehicle, ensuring the stability in both *pitch* and *roll* degrees of freedom. Its five thrusters will allow $ICTINEU^{AUV}$ to be operated in the remaining degrees of freedom (*surge*, *sway*, *heave* and *yaw*) achieving maximum speeds of 3 knots (see Fig. 5).

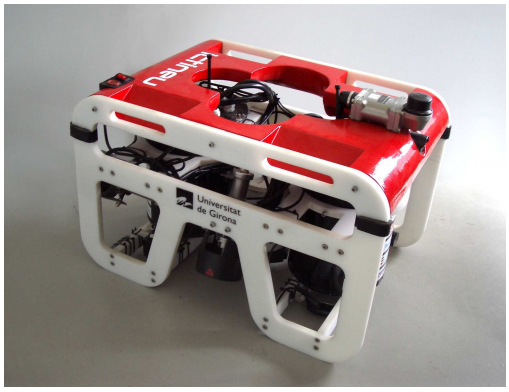


Figure 5. The autonomous underwater vehicle $ICTINEU^{AUV}$.

3.2 AUV mathematical model

As described in the literature [8], the non-linear hydrodynamic equation of motion of an underwater vehicle with 6 *degrees of freedom* (DOF), in the body fixed frame, can be conveniently expressed as:

$$\tau^B + G(\eta) - D(\upsilon^B)\upsilon^B + \tau_p = (M^B_{RB} + M_A)\dot{\upsilon}^B + ... \\ ... + (C^B_{RB}(\upsilon^B) + C_A(\upsilon^B))\upsilon^B \quad (8)$$

Where:

- υ^B is the velocity vector.
- $\dot{\upsilon}^B$ is the acceleration vector.
- $\eta = (\phi\theta\psi)^T$ are the Roll, Pith and Yaw angles.
- τ^B are the forces and moments exerted by thrusters.
- $G(\eta)$ are the gravity and buoyancy forces.
- $D(\upsilon^B)$ are the linear and quadratic damping matrixes.
- $D(\tau_P$ are the not modeled perturbations.
- M^B_{RB} is the inertia matrix.
- M^B_A is the added mass matrix.
- C^B_{RB} is the rigid body Coriolis and centripetal matrix.
- C^B_A is the hydrodynamic Coriolis and centripetal matrix.

Identification of the complete set of coefficients and hydrodynamic derivatives which appear in Eq. 8 is a rather complex task. The identification problem can be much more easily approached if the following simplifications are applied:

- $D(\upsilon^B)$ consists of the lineal and quadratic damping forces and can be assumed diagonal.
- M^B_{RB} and M^B_A can be assumed diagonal (this is true for $ICTINEU^{AUV}$ due to its squared shape, see Subsection 3.1).
- The body frame is located at the gravity center.

Moreover, if the robot is actuated in a single DOF during the identification experiments, further simplifications can be carried out. For instance, let's consider the dynamic equation for the surge (movement along X axis) DOF:

$$\tau_u + (\sin(\theta)B - \sin(\theta)W) - (X_u + X_{u|u|}|u|)u + ... \\ ... + \tau_P = (m - X_{\dot{u}})\dot{u} + [(m - Z_{\dot{w}})wq] - [(m - Y_{\dot{v}})vr] \quad (9)$$

which follows the standard notation proposed in [8]. If we excite the robot in a single DOF, surge in this case, in such a way that:

- $u \neq 0$ and $v = w = p = q = r = 0$
- $\theta = \phi = 0$

uncoupled experiment can be achieved, Eq. 9 can be rewritten as:

$$\dot{u} = -\frac{X_u}{(m - X_{\dot{u}})}u - \frac{X_{u|u|}|u|}{(m - X_{\dot{u}})}u + \frac{\tau_u}{(m - X_{\dot{u}})} + \frac{\tau_P}{(m - X_{\dot{u}})} \quad (10)$$

The same procedure can be applied to each degree of freedom so we can consider a generic uncoupled equation of motion for the i-degree of freedom as:

$$\dot{x}_i = \alpha_i x_i + \beta_i x_i|x_i| + \gamma_i\tau_i + \delta_i \quad (11)$$

where the state variable x represents speed. Hence, things become easier if we use Eq. 11 for the identification. Once the whole model has been uncoupled, its parameters have been identified by means of parameter identification methods [20]. The resultant model has been reduced to emulate a robot with only three degrees of freedom (DOF), X movement, Y movement and rotation respect Z axis. The identified parameters can be see in Table 1. Note that related to the squared shape of the vehicle, the robot behavior has been considered equal in X and Y; also, the β term corresponding to the quadratic damping has been neglected due to the low speeds achieved by $ICTINEU^{AUV}$ during the tests.

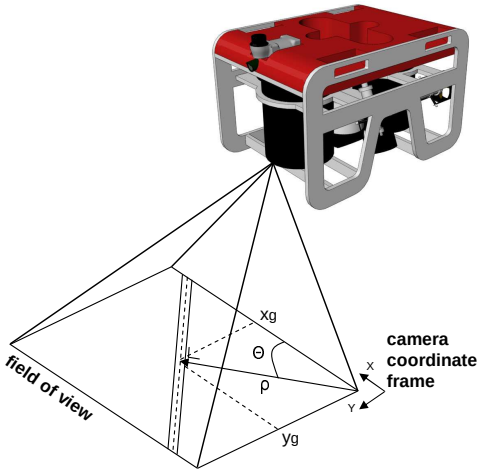


Figure 6. Coordinates of the target cable with respect $ICTINEU^{AUV}$.

Table 1. Parameter identification results.

	Parameters			
DOF	α	β	γ	δ
Surge (X movement)	0.3222	0	0.0184	0.0012
Sway (Y movement)	0.3222	0	0.0184	0.0012
Yaw (Z rotation)	1.2426	0	0.5173	-0.050

3.3 The Cable Tracking Vision System

The downward-looking B/W camera installed on $ICTINEU^{AUV}$ will be used for the vision algorithm to track the cable. It provides a large underwater field of view (about 57° in width by 43° in height). This kind of sensor will not provide us with absolute localization information but will give us relative data about position and orientation of the cable respect to our vehicle: if we are too close/far or if we should move to the left/right in order to center the object in our image. The vision-based algorithm used to locate the cable was first proposed in [17] and later improved in [3]. It exploits the fact that artificial objects present in natural environments usually have distinguishing features; in the case of the cable, given its rigidity and shape, strong alignments can be expected near its sides. The algorithm will evaluate the polar coordinates ρ (orthogonal distance from the origin of the camera coordinate frame) and Θ (angle between ρ and X axis of the camera coordinate frame) of the straight line corresponding to the detected cable in the image plane (see Fig. 6).

Once the cable has been located and the polar coordinates of the corresponding line obtained, as the cable is not a thin line but a large rectangle, we will also compute the cartesian coordinates (x_g,y_g) (see Fig. 6) of the cable's centroid with respect to the image plane by means of (12).

$$\rho = xcos(\Theta) + ysin(\Theta) \tag{12}$$

where x and y correspond to the position of any point of the line in the image plane. The computed parameters x_g and Θ together with its derivatives $\frac{\delta x_g}{\delta t}$ and $\frac{\delta \Theta}{\delta t}$ will conform the input state to our policy function represented by the barycentric interpolator. For the simulated phase, a downward-looking camera model has been used to emulate the vision system of the vehicle.

3.4 Design of the barycentric interpolator to represent the policy

As stated in previous section, the observed state is a 4 dimension vector $x = (x_g, \Theta, \frac{\delta x_g}{\delta t}, \frac{\delta \Theta}{\delta t})$. The x component of the cable centroid in the image plane x_g is ranged from $[0, 1.078]$ meters, $\Theta = [\frac{-\pi}{2}, \frac{\pi}{2}]$ radians, the derivative of $x_g, \frac{\delta x_g}{\delta t} = [-0.5, 0.5] m/s$ and finally, $\frac{\delta \Theta}{\delta t} = [-1, 1] rad/s$. From these observations, the robot will take decisions concerning two degrees of freedom, Y movement (Sway) and Z axis rotation (Yaw), therefore the continuous action vector is defined as $u = (u_{sway}, u_{yaw})$ where $u_{sway} = [-1, 1] m/s$ and $u_{yaw} = [-1, 1] rad/s$. The X movement of the vehicle (surge) will not be learned. A simple controller has been implemented to control the X DOF; only if the cable is centered in the image plane the robot will move forward ($u_{surge} = 0.3 m/s$), otherwise $u_{surge} = 0$.

In order to decrease the function approximator complexity reducing the number of grid points, the main policy has been split into two subpolicies, each one represented by a 2-dimensional barycentric interpolator. It can be easily noticed that u_{sway} actions will be mainly affected by the position and the velocity of x_g along the X axis of the image plane. In the same way, u_{yaw} actions will strongly depend on the angle Θ of the cable in the image plane. Although learning uncoupled policies will probably reduce the final performance of the learner, it has been a good initial startup to focus the problem. In Fig. 7, the observed substate (x_g, $\frac{\delta x_g}{\delta t}$) is the input of subpolicy a), being the output u_{sway}. Subpolicy b) has (Θ, $\frac{\delta \Theta}{\delta t}$) as input variables and u_{yaw} as output. The density factor δ of the barycentric mesh for both grids has been experimentally set to 10 equal divisions for each axis, therefore the mesh has 100 cells.

4 RESULTS

4.1 First phase: Simulated Learning

The model of the underwater robot $ICTINEU^{AUV}$ navigates a two dimensional world at 1 meter height above the seafloor. The simulated cable is placed at the bottom in a fixed circular position. The learner has been trained in an episodic task. An episode ends either every 15 seconds (150 iterations) or when the robot misses the cable in the image plane, whatever comes first. When the episode ends, the robot position is reset to a random position and orientation around the cable's location, assuring any location of the cable within the image plane at the beginning of each episode. According to the values of the state parameters θ and x_g, a scalar immediate reward is given each iteration step. Three values were used: -10, -1 and 0. In order to maintain the cable centered in the image plane, the non negative reward $r = 0$ is given when the position along the X axis of the centroid (x_g) is around the center of the image ($x_g \pm 0.15$) and the angle θ is close to 90° ($90^\circ \pm 15^\circ$). A $r = -1$ is given in any other location within the image plane. The reward value of -10 is given when the vehicles misses the target and the episode ends.

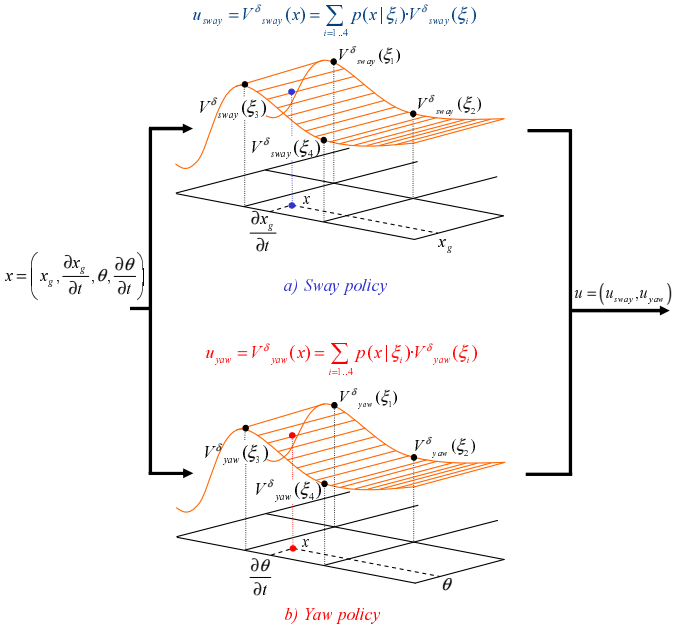


Figure 7. Representation of both subpolicies. a) represents the sway policy. b) represents the yaw policy.

The number of episodes to be done has been set to 2000. For every episode, the total amount of reward perceived is calculated. Fig. 8 represents the performance of the computed policy as a function of the number of episodes when trained using Baxter and Bartlett's algorithm. The experiment has been repeated in 100 independent runs. The results here presented are the mean over these runs. The learning rate was set to $\alpha = 0.001$ and the discount factor $\beta = 0.98$. In Fig. 9 and Fig. 10 we can observe a state/action mapping of a trained agent in both, yaw and sway degrees of freedom. Figure 11 represents the trajectory once the training period finishes.

4.2 Second phase: Learned policy transfer. Real test

Once the learning process is considered to be finished, the resultant policy is transferred to $ICTINEU^{AUV}$ and its performance tested in a real environment. The experimental setup can be seen in Fig. 12 where the detected cable is shown while the vehicle performs a test inside the pool. Fig. 13 represents real measured trajectories of the θ angle while the vehicle performs different attempts to center the cable in the image.

5 CONCLUSIONS AND FUTURE WORKS

This paper proposes a field application of a high-level Reinforcement Learning (RL) control system for solving the action selection problem of an autonomous robot in cable tracking task. The learning system is characterized by using a direct policy search algorithm for robot control based on Baxter and Bartlett's direct-gradient algorithm. The policy is represented by 2 barycentric interpolators with

2 input state variables, each one controlling one degree of freedom. In order to speed up the process, the learning phase has been carried out in a simulated environment and then transferred and tested successfully on the real robot $ICTINEU^{AUV}$.

Results of this work show a good performance of the learned policy. Although it is not a hard task to learn in simulation, continuing the learning autonomously in a real situation represents a challenge due to the nature of underwater environments. We believe that direct policy search reinforcement learning methods can be a good solution to solve decision making problems in such a hostile situation as underwater missions. Future steps are focused on continue to improve the policy by means of on-line learning in real environment and comparing the results obtained with human pilots tracking tra-

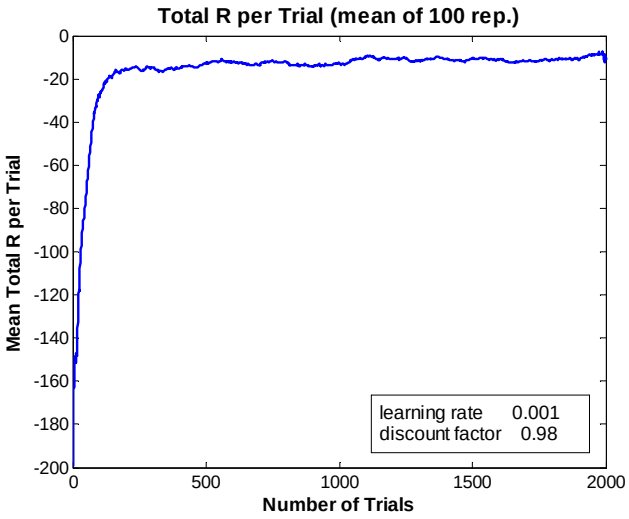


Figure 8. Performance of the neural-network robot controller as a function of the number of episodes. Performance estimates were generated by simulating 2000 episodes. Process repeated in 100 independent runs. The results are a mean of these runs. Fixed $\alpha = 0.001$, and $\beta = 0.98$.

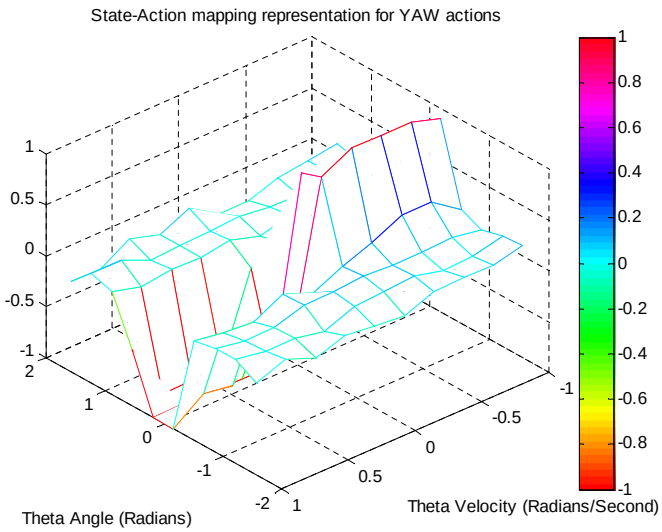


Figure 9. Theta angle - Theta velocity mapping of a trained robot controller. Colorbar on the right represents the actions taken.

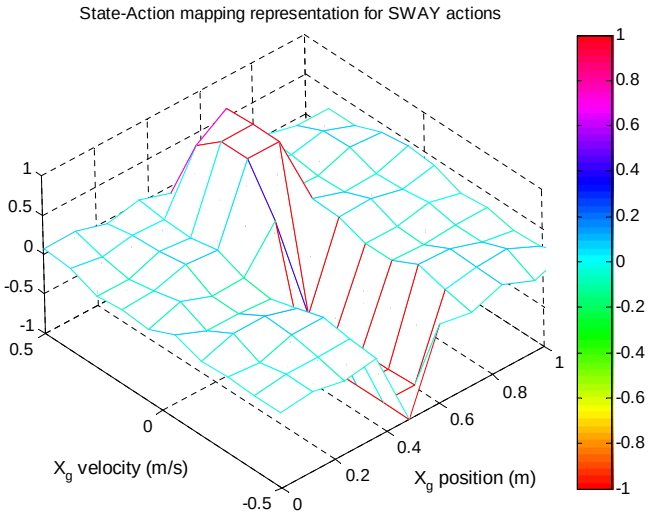


Figure 10. Centroid X position - Centroid X velocity mapping of a trained robot controller. Colorbar on the right represents the actions taken.

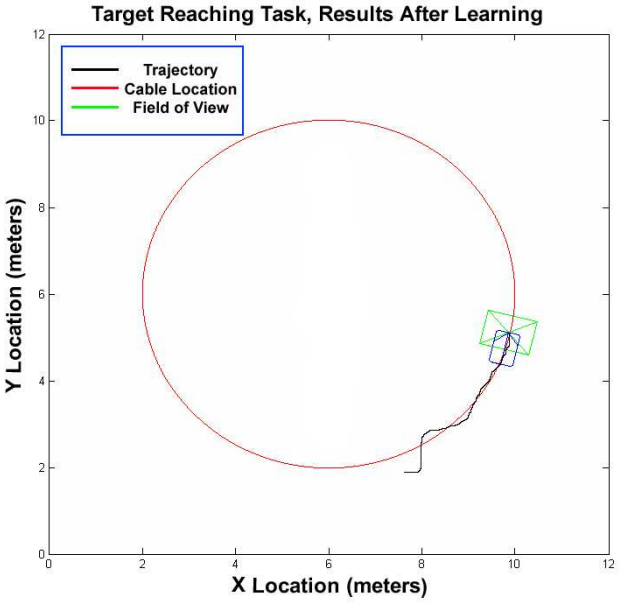


Figure 11. Behavior of a trained robot controller, results of the simulated cable tracking task after learning period is completed.

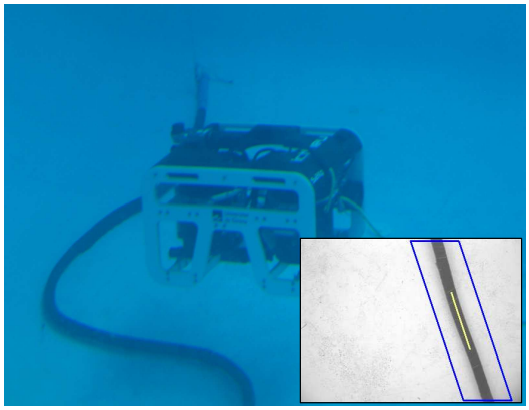


Figure 12. $ICTINEU^{AUV}$ in the test pool. Small bottom-right image: Detected cable.

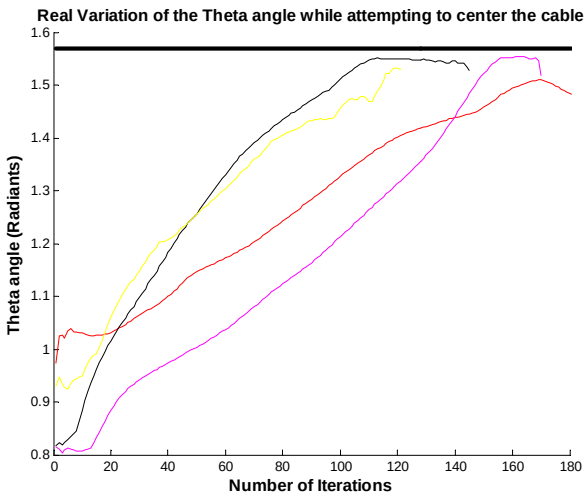


Figure 13. 4 real measured trajectories of the θ angle of the image plane while attempting to center the cable at $\pi/2$ rad. Every iteration represents 0.1 seconds.

jectories and other controllers.

ACKNOWLEDGEMENTS

First of all, we would like to give our special thanks to the group of the University of the Balearic Islands for allowing us to use their cable detection algorithm. This work has been financed by the Spanish Government Comission MCYT, project number DPI2005-09001-C03-01, also partially funded by the MOMARNET EU project MRTN-CT-2004-505026 and the European Research Training Network on Key Technologies for Intervention Autonomous Underwater Vehicles FREESUBNET, contract number MRTN-CT-2006-036186.

REFERENCES

[1] D. A. Aberdeen, *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*, Ph.D. dissertation, Australian National University, April 2003.
[2] C. Anderson, 'Approximating a policy can be easier than approximating a value function', Computer science technical report, University of Colorado State, (2000).
[3] J. Antich and A. Ortiz, 'Underwater cable tracking by visual feedback', in *First Iberian Conference on Pattern recognition and Image Analysis (IbPRIA, LNCS 2652)*, Port d'Andratx, Spain, (2003).
[4] C.G. Atkenson, A.W. Moore, and S. Schaal, 'Locally weighted learning', *Artificial Intelligence Review*, **11**, 11–73, (1997).
[5] J.A. Bagnell and J.G. Schneider, 'Autonomous helicopter control using reinforcement learning policy search methods', in *Proceedings of the IEEE International Conference on Robotics and Automation*, Korea, (2001).
[6] J. Baxter and P. Bartlett, 'Direct gradient-based reinforcement learning: I. gradient estimation algorithms', Technical report, Australian National University, (1999).
[7] A. El-Fakdi, M. Carreras, and P. Ridao, 'Towards direct policy search reinforcement learning for robot control', in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2006).
[8] T. I. Fossen, *Guidance and Control of Ocean Vehicles*, John Wiley and Sons, 1995.
[9] Bradley Hammer, Sanjiv Singh, and Sebastian Scherer, 'Learning obstacle avoidance parameters from operator behavior', *Journal of Field Robotics, Special Issue on Machine Learning Based Robotics in Unstructured Environments*, **23 (11/12)**, (December 2006).
[10] N. Kohl and P. Stone, 'Policy gradient reinforcement learning for fast quadrupedal locomotion', in *IEEE International Conference on Robotics and Automation (ICRA)*, (2004).
[11] V.R. Konda and J.N. Tsitsiklis, 'On actor-critic algorithms', *SIAM Journal on Control and Optimization*, **42, number 4**, 1143–1166, (2003).
[12] L.J. Lin, 'Self-improving reactive agents based on reinforcement learning, planning and teaching.', *Machine Learning*, **8(3/4)**, 293–321, (1992).
[13] P. Marbach and J. N. Tsitsiklis, 'Gradient-based optimization of Markov reward processes: Practical variants', Technical report, Center for Communications Systems Research, University of Cambridge, (March 2000).
[14] T. Matsubara, J. Morimoto, J. Nakanishi, M. Sato, and K. Doya, 'Learning sensory feedback to CPG with policy gradient for biped locomotion', in *Proceedings of the International Conference on Robotics and Automation ICRA*, Barcelona, Spain, (April 2005).
[15] N. Meuleau, L. Peshkin, and K. Kim, 'Exploration in gradient based reinforcement learning', Technical report, Massachusetts Institute of Technology, AI Memo 2001-003, (April 2001).
[16] R. Munos and A. Moore, 'Barycentric interpolators for continuous space and time reinforcement learning', (1998).
[17] A. Ortiz, M. Simo, and G. Oliver, 'A vision system for an underwater cable tracker', *International Journal of Machine Vision and Applications*, **13 (3)**, 129–140, (2002).
[18] J. Peters and S. Schaal, 'Policy gradient methods for robotics', in *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS'06*, Beijing, China, (October 9-15 2006).
[19] D. Ribas, N. Palomeras, P. Ridao, M. Carreras, and E. Hernandez, 'Ictineu auv wins the first sauc-e competition', in *IEEE International Conference on Robotics and Automation*, (2007).

[20] P. Ridao, A. Tiano, A. El-Fakdi, M. Carreras, and A. Zirilli, 'On the identification of non-linear models of unmanned underwater vehicles', *Control Engineering Practice*, **12**, 1483–1499, (2004).
[21] M.T. Rosenstein and A.G. Barto, 'Robot weightlifting by direct policy search', in *Proceedings of the International Joint Conference on Artificial Intelligence*, (2001).
[22] W.D. Smart, *Making Reinforcement Learning Work on Real Robots*, Ph.D. dissertation, Department of Computer Science at Brown University, Rhode Island, May 2002.
[23] R. Sutton and A. Barto, *Reinforcement Learning, an introduction*, MIT Press, 1998.
[24] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour, 'Policy gradient methods for reinforcement learning with function approximation', *Advances in Neural Information Processing Systems*, **12**, 1057–1063, (2000).
[25] R. Tedrake, T. W. Zhang, and H. S. Seung, 'Stochastic policy gradient reinforcement learning on a simple 3D biped', in *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS'04*, Sendai, Japan, (September 28 - October 2 2004).

1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2008)
Patras, Greece, July 22 2008

Incremental Basis Function Expansion in Reinforcement Learning using Cascade-Correlation Networks

Sertan Girgin[1] and **Philippe Preux**[1,2]

Abstract. In machine learning, in parallel to algorithms themselves, the representation of data is a point of utmost importance. Efforts on data pre-processing in general are a key ingredient to success. An algorithm that performs poorly on a particular form of given data may perform much better, both in terms of efficiency and the quality of the solution, when the same data is represented in another form. Despite the amount of literature on the subject, the issue of how to enrich a representation to suit the underlying mechanism is clearly still pending. In this paper, we approach this problem within the context of reinforcement learning, and in particular, interested in discovery of a "good" representation of data for the LSPI algorithm. To this end, we use the cascade-correlation learning architecture to automatically generate a set of basis functions which would lead to a better approximation of the value function, and consequently improve the performance of the resulting policies. This is especially important within the context of learning in autonomus robot systems, as manually determining an effective set of basis functions generally requires an in-depth understanding of the complex problem domain. We show the effectiveness of the idea on some benchmark problems.

1 Introduction

Reinforcement learning (RL) is the problem faced by an agent that is situated in an environment and must learn a particular behavior through repeated trial-and-error interactions with it [20]; at each time step, the agent observes the state of the environment, chooses its action based on these observations and in return receives some kind of "reward", in other words a *reinforcement signal*, from the environment as feedback. The aim of the agent is to find a policy, a way of choosing actions, that maximizes its overall gain – a function of rewards, such as the (discounted) sum or average over a time period. RL has been and is being extensively studied under different settings (online / offline, discrete / continuous state-action spaces, discounted / average reward, perfect / imperfect state information; etc.) and various methods and algorithms (dynamic programming, Monte Carlo methods, temporal-difference learning, etc.) have been proposed. One common point to all such approaches is that, regardless of how they do it what is being produced as a solution is a mapping from *inputs*, observations, to *outputs*, actions. It is natural that different approaches may require or prefer the input data be in different forms, but still the same data can be represented in many different ways conforming to the specified form. This brings up the questions of what the best representation of input data is for a given method or algorithm, and how it can be found. This point is a major issue, not only in reinforcement learning, but also in machine learning, and even, in artificial intelligence, and computer science.

In particular, in this paper, we will focus on Least-Squares Policy Iteration (LSPI) algorithm [10] which uses a linear combination of basis functions to approximate a state-action value function and learn a policy from a given set of experience samples. The basis functions map state-action tuples into real numbers, and each basis function performs a mapping that is different from the others. From the point of view of LSPI, the real input becomes the values of basis functions evaluated for given state-action pairs. Therefore, the set of basis functions that is being employed directly affects the quality of the solution, and the question transforms into "what is the set of best basis functions?". We seek to provide a possible answer to this question by proposing a method that incorporates a cascade correlation learning architecture into LSPI and iteratively adds new basis functions to a set of initial basis functions. In Section 2, we first introduce policy iteration and the LSPI algorithm. Sections 3 describes cascade correlation learning architecture followed by the details of the proposed method for basis function expansion in Section 4. Section 5 presents empirical evaluations on some benchmark problems, and a review of related work can be found in Section 6. Finally, Section 7 concludes.

2 Least-Squares Policy Iteration

2.1 Markov Decision Problems

In this paper, we assume that the time t flows in a discrete manner, that is $t \in \mathbb{N}$. We consider the situation where an agent repeatedly perceives the state of its environment, acts on it, which leads to a next state and an immediate reward, in order to optimize a certain reward function along time.

Following [17], a *Markov decision problem* (MDP) is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{S}$ is a set (finite, or not) of states,
- $\mathcal{A}$ is a set of actions,
- $\mathcal{P}(s, a, s')$ is the transition function which denotes the probability of making a transition from state s to state s' by taking action a, that is $Pr[s_{t+1} = s' | s_t = s, a_t = a] = \mathcal{P}(s, a, s')$,
- $\mathcal{R}(s, a)$ is the expected (immediate) reward function, that is the expected immediate reward at time t, $\mathbb{E}[r_t] = \mathcal{R}(s_t, a_t)$,

In this paper, the function to be maximized is the expected total discounted reward from any state s, that is:

$$R(s) = \sum_{t \geq 0} \gamma^t r_t(s_t, a_t) | s_0 = s$$

where $\gamma \in [0, 1)$ is the discount factor that determines the importance of future rewards.

[1] Team-Project SequeL, INRIA Lille Nord-Europe
[2] LIFL (UMR CNRS), Université de Lille
email: {sertan.girgin, philippe.preux}@inria.fr

A *policy* is a probability distribution over actions conditioned on the state; $\pi(s,a)$ denotes the probability that policy π selects action a at state s. An optimal policy maximizes the reward function R for any initial state if followed.

Policy iteration is a general framework for finding an optimal policy; starting from an initial policy, two basic steps, namely *policy evaluation* followed by *policy improvement*, are applied consecutively and iteratively until convergence to an optimal policy is achieved or a certain stopping criterion is met.

Let π_i be the policy at iteration i. A policy evaluation step consists of finding the state value function,

$$V^{\pi_i}(s) = \mathbb{E}_{a_t \sim \pi_i}\big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\big]$$

or the state-action value function,

$$Q^{\pi_i}(s,a) = \mathbb{E}_{a_t \sim \pi_i, t>0}\big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\big]$$

of the current policy π_i, where a_t is the action chosen according to π_i and r_t is the reward received at time t. In some restricted cases[3], this can be accomplished by solving numerically or analytically the system of the Bellman equations:

$$Q^{\pi_i}(s,a) = \mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s,a,s') V^{\pi_i}(s') ds' \quad (1)$$

$$V^{\pi_i}(s) = \int_{\mathcal{A}} \pi_i(s,a) Q^{\pi_i}(s,a) da \quad (2)$$

The integrals over state and action spaces are replaced by finite sums in the discrete case. The right-hand side of the equations applied to a given state(-action) value function for a policy π define the *Bellman operators* denoted by T_π. Alternatively, one can employ *Monte Carlo methods* by sampling multiple trajectories (i.e. a sequence $s_0 a_0 r_0 s_1 \ldots$ of states, actions and rewards) and evaluate the expectations by taking the average value over such roll-outs, or use *temporal difference learning* in which the Bellman equations are considered as update rules and state(-action) value function estimate is successively updated based on the previous estimates.

In the policy improvement step, the state(-action) value function obtained in the policy evaluation step is used to derive a new policy π_{i+1} which would perform at least as well as π_i, i.e. satisfies $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$ for all $s \in \mathcal{S}$. For a deterministic policy, this can be realized by defining π_{i+1} greedy with respect to Q^{π_i} as

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi_i}(s,a) \quad (3)$$

Policy iteration is guaranteed to converge to an optimal policy in a finite number of steps if both state and action spaces are finite, the value function and the policy are represented perfectly and the policy evaluation step is solved exactly. However, in most cases it may not be possible to fulfill these requirements (eg. in problems with large or infinite state and action spaces) and the value function and/or the policy need to be approximated, leading to the so-called *approximate policy iteration* approach. It has been shown that if the error in the approximations are bounded then approximate policy iteration generates policies such that their performance is also bounded with respect to the optimal policy, yet the performance bound can be arbitrarily large as the discount factor, γ, gets closer to 1 [2, 15].

[3] Such as problems with finite and small state-action spaces, or linear quadratic optimal control problems in which the underlying MDP model is known.

2.2 Least-Squares Policy Iteration

Least-Squares policy iteration (LSPI) is an off-line and off-policy approximate policy iteration algorithm proposed by Lagoudakis and Parr (2003). Rather than relying on continual interactions with the environment or a generative model, it works on a fixed set of samples collected arbitrarily. Each sample is of the form (s,a,r,s') indicating that executing action a at state s resulted in a transition to state s' with an immediate reward of r. The state-action value function is approximated by a linear form

$$\widehat{Q}^{\pi}(s,a) = \sum_{j=0}^{m-1} w_j \phi_j(s,a)$$

where $\phi_j(s,a)$ denote the *basis functions* and $w_j \in \mathbb{R}$ are the parameters. Basis functions, also called *features*, are arbitrary functions of state-action pairs, but are intended to capture the underlying structure of the target function and can be viewed as doing dimensionality reduction from a larger space to $\Re^m$. Typical examples of basis functions are polynomials of a given degree or radial basis functions, such as Gaussian and multi-quadratics, each possibly associated with a different center and scale. Note that, in the case of discrete state-action spaces, this generic form also includes tabular representations as a special case, but in general the number of basis functions, m, is much smaller compared to $|\mathcal{S}|\,|\mathcal{A}|$.

Instead of representing the policy explicitly, LSPI opts to determine the action that is imposed by the current policy at a given state by directly evaluating Eq. 3 (hence the policy improvement step becomes inherent). This may not be a feasible operation when the number of actions is large or possibly infinite, except in certain cases where a closed form solution is achievable; however, in many situations, this drawback may not pose a significant problem as the action space is generally less susceptible to discretization compared to the state space.

Given a policy π_{i+1}, greedy with respect to $\widehat{Q}^{\pi_i}$ which is defined by the set of parameters $w_j^{\pi_i}$, LSPI performs the policy evaluation step and determines $\widehat{Q}^{\pi_{i+1}}$, in other words the corresponding set of new parameters $w_j^{\pi_{i+1}}$, by invoking an algorithm called LSTDQ. One can easily observe that by definition the state-action value function Q^π of a policy π is necessarily a fixed point of the Bellman operator, i.e. $Q^\pi = T_\pi Q^\pi$ (Eq. (1)), which also holds for $Q^{\pi_{i+1}}$. Due to the specific choice of linear function approximation, any $\widehat{Q}^\pi$ is confined to the subspace spanned by the basis functions $\{\phi_{j\in\{1,\ldots,m\}}\}$, and therefore, an approximation $\widehat{Q}^{\pi_{i+1}}$ to $Q^{\pi_{i+1}}$ which stays invariant under the Bellman operator may not exist. Instead, LSTDQ tries to find an approximation $\widehat{Q}^{\pi_{i+1}}$ which is equal to the orthogonal projection of its image under the Bellman operator. Such $\widehat{Q}^{\pi_{i+1}}$ also possesses the property that

$$\widehat{Q}^{\pi_{i+1}} = \arg\min_{\widehat{Q}^\pi} \| T_{\pi_{i+1}} \widehat{Q}^{\pi_{i+1}} - \widehat{Q}^\pi \|_2$$

A motivation for choosing this particular approximation is expressed as the expectation that the approximation should be close to the projection of $Q^{\pi_{i+1}}$ onto the subspace spanned by the basis functions if subsequent applications of the Bellman operator point in a similar direction. Without going into the details, given any N samples, LSTDQ finds $\widehat{Q}^{\pi_{i+1}}$ by solving the $m \times m$ system

$$\widetilde{A} w^{\pi_{i+1}} = \widetilde{b}$$

where $\phi(s,a) = [\phi_0(s,a)\phi_1(s,a)\ldots\phi_{m-1}(s,a)]^\top$ is the row vec-

Algorithm 1 The LSPI algorithm.

Require: $\mathcal{S}$: Set of samples, $\vec{\phi}$: basis functions, γ: discount factor.
1: **function** LSTDQ($\mathcal{S}, \vec{\phi}, \gamma, \pi_w$) ▷ π_w is the policy parameterized by w.
2: $\tilde{A} \leftarrow \mathbf{0}, \tilde{\vec{b}} \leftarrow \mathbf{0}$ ▷ $\tilde{A}$ is a $|\vec{\phi}| \times |\vec{\phi}|$ matrix and $\tilde{\vec{b}}$ is a $|\vec{\phi}| \times 1$ vector.
3: **for each** $(s, a, r, s') \in \mathcal{S}$ **do** ▷ Iterate over the sample set
4: $\tilde{A} \leftarrow \tilde{A} + \phi(s,a)\left[\phi(s,a) - \gamma\phi(s', \pi_w(s'))\right]^T$
5: $\tilde{\vec{b}} \leftarrow \tilde{\vec{b}} + \phi(s,a)r$
6: **end for**
7: **return** $\tilde{A}^{-1}\tilde{\vec{b}}$ ▷ parameters of the new policy
8: **end function**
9:
10: **function** LSPI($\mathcal{S}, \vec{\phi}, \gamma$)
11: $i \leftarrow 0$
12: $\vec{w}_0 \leftarrow$ initial weights
13: **repeat** ▷ Update the policy until it converges.
14: $i \leftarrow i + 1$
15: $\vec{w}_i \leftarrow \texttt{LSTDQ}(\mathcal{S}, \vec{\phi}, \gamma, \pi_{w_{i-1}})$
16: **until** $\|w_i - w_{i-1}\| < \epsilon$ or $i > i_{max}$ ▷ ϵ is the accuracy threshold.
17: **return** w_i
18: **end function**

tor of basis functions evaluated at (s,a), and

$$\widetilde{A} = \frac{1}{N}\sum_{i=1}^{N}\left[\phi(s_i,a_i)\Big(\phi(s_i,a_i) - \gamma\phi(s_i', \pi_{i+1}(s_i'))\Big)^\top\right],$$

$$\widetilde{b} = \frac{1}{N}\sum_{i=1}^{N}\phi(s_i,a_i)r_i,$$

both of which in the limit converge to the matrices of the least-squares fixed-point approximation obtained by replacing $\widehat{Q}^{\pi_{i+1}} = \Phi w^{\pi_{i+1}}$ in the system

$$\widehat{Q}^{\pi_{i+1}} = \Phi(\Phi^\top\Phi)^{-1}\Phi^\top(T_{\pi_{i+1}}\widehat{Q}^{\pi_{i+1}})$$

Here, $\Phi(\Phi^\top\Phi)^{-1}\Phi^\top$ is the orthogonal projection and Φ denotes the matrix of the values of the basis functions evaluated for the state-action pairs. The details of the derivation can be found in the seminal paper [10].

The LSPI algorithm presented in Algorithm 1 has been demonstrated to provide "good" policies within relatively small number of iterations. Furthermore, as it is possible to use a single and common sample set for all policy evaluations, LSPI makes efficient use of the available data; as such, it is quite suitable for problems in which data gathering process is time consuming and costly. However, the quality of the resulting policies depends on two important factors: the basis functions, and the distribution of the samples.

In an off-line setting, one may not have any control on the set of samples and too much bias in the samples would inevitably reduce the performance of the learned policy. On the other hand, in an on-line setting, LSPI allows different sample sets to be employed at each iteration; thus, it is possible to fine tune the trade-off between exploration and exploitation by collecting new samples using the current policy.

Regarding the basis functions, the user is free to choose any set of functions as long as they are linearly independent (a restriction which can be relaxed in most cases by applying singular value decomposition). As shown in [10], and in accordance with the generic performance bound on policy iteration, if the error between the approximate and the true state-action value functions at each iteration is bounded by a scalar ϵ, then in the limit the error between the optimal state-action value function and those corresponding to the policies generated by LSPI is also bounded by a constant multiple of ϵ. Therefore, selecting a "good" set of basis functions has a significant and direct effect on the success of the method.

In general, the set of basis functions is defined by the user based on domain knowledge, and usually in a trial and error fashion. They can either be fixed, or one can start from an initial subset of predefined basis functions and iteratively introduce remaining functions based on the performance of the current set, so-called *feature iteration approach* [1]. However, as the complexity of the problem increases it also gets progressively more difficult to come up with a good set of basis functions. Generic approaches, such as regular grids or regular radial basis function networks, which are quite successful in small problems, become impractical due to the exponential growth of the state-action spaces with respect to their dimension. Therefore, given a problem, it is highly desirable to determine a compact set of such basis functions automatically. In the next section, we will first describe a particular class of function approximators called *cascade-correlation networks*, and then present how they can be utilized in the LSPI algorithm to iteratively expand the set of basis functions.

3 Cascade Correlation Networks

Cascade correlation is both an architecture and a supervised learning algorithm for artificial neural networks introduced by [4]. It aims to overcome *step-size* and *moving target* problems that negatively affect the performance of back-propagation learning algorithm. Similar to traditional neural networks, the neuron is the most basic unit in cascade correlation networks. However, instead of having a predefined topology with the weights of the fixed connections between neurons getting adjusted, a cascade correlation network starts with a minimal structure consisting only of an input and an output layer, without any hidden layer. All input neurons are directly connected to the output neurons (Figure 1a). Then, the following steps are taken:

1. All connections leading to output neurons are trained on a sample set and corresponding weights (i.e. only the input weights of output neurons) are determined by using an ordinary learning algorithm until the error of the network no longer decreases. This can be done by applying the regular "delta" rule, or using more efficient methods such as quick-prop or RPROP. Note that, only the input weights of output neurons (or equivalently the output weights of input neurons) are being trained, therefore there is no back-propagation.
2. If the accuracy of the network is above a given threshold then the process terminates.
3. Otherwise, a set of *candidate units* is created. These units typically have non-linear activation functions, such as sigmoid or Gaussian. Every candidate unit is connected with all input neurons and with all existing hidden neurons (which is initially empty); the weights of these connections are initialized randomly. At this stage the candidate units are not connected to the output neurons, and therefore are not actually active in the network. Let s denote a training sample. The connections leading to a candidate unit are trained with the goal of maximizing the sum $\mathcal{S}$ over all output units o of the magnitude of the correlation between the candidate units value denoted by v_s, and the residual error observed at output neuron o

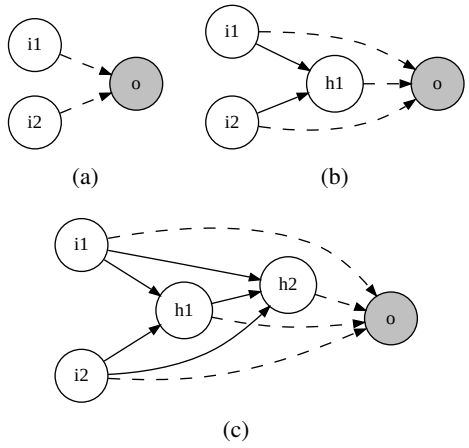


Figure 1. (a) Initial configuration of a simple cascade-correlation network with two inputs and a single output (in gray). (b) and (c) show the change in the structure of the network as two new hidden nodes are subsequently added. Solid edges indicate input weights that stay fixed for ever, after the candidate training phase, whereas dashed edges will be further trained: the edges undergoing this ongoing weight training are precisely those that are connected to the inputs of the output unit.

denoted by $e_{s,o}$. S is defined as

$$S = \sum_o |\sum_s (v_s - v)(e_{s,o} - e_o)|$$

where v and e_o are the values of v_s and $e_{s,o}$ averaged over all samples, respectively. As in step 1, learning takes place with an ordinary learning algorithm by performing gradient ascent with respect to each of the candidate units incoming weights:

$$\frac{\partial S}{\partial w_i} = \sum_{s,o} (e_{s,o} - e_o)\sigma_o f'_s I_{i,s}$$

where σ_o is the sign of the correlation between the candidates value and output o, f'_s is the derivative for sample s of the candidate units activation function with respect to the sum of its inputs, and $I_{i,s}$ is the input the candidate unit received from neuron i for sample s. Note that, since only the input weights of candidate units are being trained there is again no need for back-propagation. Besides, it is also possible to train candidate units in parallel since they are not connected to each other. By training multiple candidate units instead of a single one, different parts of the weight space can be explored simultaneously. This consequently increases the probability of finding neurons that are highly correlated with the residual error. The learning of candidate unit connections stops when the correlation scores no longer improve or after a certain number of passes over the training set. Now, the candidate unit with the maximum correlation is chosen, its incoming weights are frozen (i.e. they are not updated in the subsequent steps) and it is added permanently to the network by connecting it to all output neurons (Figure 1b and c). The initial weights of these connections are determined based on the value of correlation of the unit. All other candidate units are discarded.

4. Return back to step 1.

Until the desired accuracy is achieved at step 2, or the number of neurons reaches a given maximum limit, a cascade correlation network completely self-organizes itself and grows as necessary. One can easily observe that, by adding hidden neurons one at a time and freezing their input weights, training of both the input weights of output neurons (step 1) and the input weights of candidate units (step 3) reduce to one step learning problems. Since there is no error to back-propagate to previous layers the moving target problem is effectively eliminated. Also, by training candidate nodes with different activation functions and choosing the best among them, it is possible to build a more compact network that better fits the training data.

One observation here is that, unless any of the neurons has a stochastic activation function, the output of a neuron stays constant for a given sample input. This brings the possibility of storing the output values of neurons which in return reduces the number of calculations in the network and improve the efficiency drastically compared to traditional multi-layer back-propagation networks, especially for large data sets. But more importantly, *each hidden neuron effectively becomes a permanent feature detector*, or to put it another way, a basis function in the network; the successive addition of hidden neurons in a cascaded manner allows, and further, facilitates the creation of more complex feature detectors that helps to reduce the error and better represent the functional dependency between input and output values. We would like to point out that, this entire process does not require any user intervention and is well-matched to our primary goal of determining a set of good basis functions for function approximation is RL, in particular within the scope of LSPI algorithm. We will now describe our approach for realizing this.

4 Using Cascade Correlation Learning Architecture in LSPI

As described in Section 2, in LSPI the state-action value function of a given policy is approximated by a linear combination of basis functions. Our aim here is to employ cascade correlation networks as function approximators and at the same time use them to find useful basis functions in LSPI. Given a reinforcement learning problem, suppose that we have a set of state-action basis functions $\Phi = \{\phi_1(s,a), \phi_2(s,a), \ldots, \phi_m(s,a)\}$. Using this set of basis functions and applying LSPI algorithm on a set of collected samples of the form (s, a, r, s'), we can find a set of parameters $\{w_i\}_{i\in\{1,\ldots,m\}}$ together with an approximate state-action value function

$$\widehat{Q}(s,a) = \sum_{i=1}^{m} w_i \phi_i(s,a)$$

and derive a policy $\widehat{\pi}$ which is greedy with respect to $\widehat{Q}$. Let $\mathcal{N}$ be a cascade correlation network with m inputs and a single output having linear activation function. In this case, the output of the network is a linear combination of the activation values of input and hidden neurons of the network weighted by their connection weights. Initially, the network does not have any hidden neurons and all input neurons are directly connected to the output neuron. Therefore, by setting the activation function of the i^{th} input neuron to ϕ_i and the weight of its connection to the output neuron to w_i, $\mathcal{N}$ becomes functionally equivalent to $\widehat{Q}$ and outputs $\widehat{Q}(s,a)$ when input neurons receive the (s,a) tuple as their input.

Now, the Bellman operator T_π is known to be a contraction in L_∞ norm, that is for any state-action value function Q, $T_\pi Q$ is closer to Q^π in the L_∞ norm, and in particular as mentioned in Section 2, Q^π is a fixed point of T_π. Ideally, a good approximation would be close to its image under the Bellman operator. As opposed to the Bellman residual minimizing approximation, least-squares fixed-point approximation, which is at the core of the LSPI algorithm, ignores

the distance between $T_{\widehat{\pi}}\widehat{Q}$ and $\widehat{Q}$ but rather focuses on the direction of the change. Note that, if the true state-action value function Q^{π} lies in the subspace spanned by the basis functions, that is the set of basis functions is "rich" enough, fixed-point approximation would be solving the Bellman equation and the solution would also minimize the magnitude of the change. This hints that, within the scope of LSPI, one possible way to drive the search towards solutions that satisfy this property could be to expand the set of basis functions by adding new basis functions that are likely to reduce the distance between the found state-action value function $\widehat{Q}$ and $T_{\widehat{\pi}}\widehat{Q}$ over the sample set.

For this purpose, given a sample (s, a, r, s'), in the cascade correlation network we can set $r + \gamma\widehat{Q}(s', \widehat{\pi}(s'))$ as the target value for (s, a) tuple, and train candidate units that are highly correlated with the residual output error, i.e. $\widehat{Q}(s, a) - \Big(r + \gamma\widehat{Q}(s', \widehat{\pi}(s'))\Big)$. At the end of the training phase, the candidate unit having the maximum correlation is added to the network by transforming it into a hidden neuron, and becomes the new basis function ϕ_{m+1}; $\phi_{m+1}(s, a)$ can be calculated by feeding (s, a) as input to the network and determining the activation value of the hidden neuron. Through another round of LSPI learning, one can obtain a new least-squares fixed-point approximation to the state-action value function $\widehat{Q}'(s, a) = \sum_{i=1}^{m+1} w'_i\phi_i(s, a)$ which is more likely to be a better approximation also in the sense of Bellman residual minimization. The network is then updated by setting the weights of connections leading to the output neuron to w'_i for each basis function. This process can be repeated, introducing a new basis function at each iteration, until the error falls below a certain threshold, or a policy with adequate performance is obtained.

We can regard this as a hybrid learning system, in which the weights of connections leading to the output neuron of the cascade correlation network are being regulated by the LSPI algorithm. Note that, the values of all basis functions for a given (s, a) tuple can be found with a feed-forward run over the network, and as stated before can be cached for efficiency reasons if desired. The complete algorithm that incorporates the cascade correlation network and basis function expansion to LSPI is presented in Algorithm 2.

Algorithm 2 The LSPI algorithm with basis function expansion using cascade correlation network.

Require: $\mathcal{S}$: Set of samples, $\vec{\phi}_{initial}$: initial basis functions, γ: discount factor, n: number of candidate units.

1: Create a cascade correlation network $\mathcal{N}$ with $|\vec{\phi}_{initial}|$ inputs and a single output, and set activation functions of input unit i to $\phi_i, \forall i \in \{1, ..., n\}$.
2: $\vec{\phi} \leftarrow \vec{\phi}_{initial}$
3: **repeat**
4: $w \leftarrow LSPI(\mathcal{S}, \vec{\phi}, \gamma)$
5: Set the weight of connection between the i^{th} unit and the output in $\mathcal{N}$ to w_i.
6: Calculate the residual error, $\widehat{Q}(s, a) - (r + \gamma\widehat{Q}(s', \pi_w(s')))$, over $\mathcal{S}$.
7: Train n candidate units on $\mathcal{N}$.
8: $\kappa \leftarrow$ Candidate unit having the maximum correlation with the residual error.
9: Add κ to $\mathcal{N}$.
10: Add ϕ_κ to $\vec{\phi}$ ▷ ϕ_κ is the function represented by κ.
11: **until** termination condition is satisfied
12: **return** w and ϕ

One possible problem that may emerge with the proposed method is that, especially when the sample set is small, with increasingly complex basis functions there may be over-fitting, and the on-line performance of the resulting policy may degrade. This can be avoided by increasing the amount of samples, or alternatively a cross-validation approach can be ensued. Suppose that for a particular reinforcement learning problem, we are given multiple sample sets. The intuition is that a set of "good" basis functions should give rise to "good" policies and similar value functions for all sample sets. By applying LSPI algorithm independently on each sample set but training a single set of candidate units over all sample sets, in other words having a common set of basis functions, one can obtain basis functions, and consequently policies, that are less biased to training data.

5 Experiments

We have evaluated the proposed method on three problems: chain walk [10], pendulum swing-up and multi-segment swimmer [3].

Chain walk is an MDP consisting of a chain of n states. There are two actions, $left$ and $right$, which succeed with probability 0.9, or else fail, moving to the state in the opposite direction. The first and last states are dead-ends, i.e. going left at the first state, or right at the last state revert back to the same state. The reward is 1 if an action ends up in a predefined set of states, and 0 otherwise.

The pendulum swing-up and multi-segment swimmer problems are dynamical systems where the state is defined by the position and velocity of the elements of the system, and actions (applied forces) define the next state. These are non-linear control tasks with continuous state spaces.

In pendulum, the aim is to keep a simple pendulum actuated by a bounded torque in vertical upright position. Since the torque available is not sufficient, the controller has to swing the pendulum back and forth to reach the goal position. The state variables are the angle of the pendulum and its angular speed. We used two discrete actions, applying a torque of -5, and 5. The reward is defined as the cosine of the angle of the pole. We follow the definition of this task given in [3].

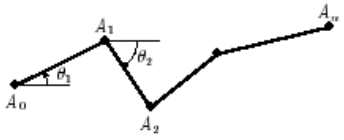


Figure 2. The swimmer is actually a multi-segmented body. One of the extremities is considered the head of the swimmer. Each segment is rigid. Forces applied at each joint make the whole body move. A simplistic fluid dynamics is simulated. The goal is to to swim as fast as possible in the horizontal direction. Commonly investigated swimmers are made of 3-7 segments.

In *swimmer*, a somewhat "idealized" swimmer (see Fig. 2) moving in a two dimensional pool is being simulated. The swimmer is made of n ($n \geq 3$) segments connected to each other with $n - 1$ joints. The goal is to swim as fast as possible to the right by applying torques to these joints and using the friction of the water. There are $2n + 2$ state variables consisting of (i) horizontal and vertical velocities of the center of mass of the swimmer, (ii) n angles of its segments with respect to vertical axis and (iii) their derivatives with

respect to time; the actions are the $n-1$ torques applied at segment joints. The reward is equal to the horizontal velocity of the swimmer. The system dynamics and more detailed information about Swimmer problem can be found in [3].

In all problems, we started from a set of basis functions consisting of the following:

1. a constant bias function (i.e. 1),
2. a basis function for each one of the state variables, which returns the normalized value of that variable, and
3. a basis function for each possible value of each control variable, which returns 1 if the corresponding control variable in the state-action tuple is equal to that value, and 0 otherwise.

Therefore, the number of the initial basis functions were 4 (1+1+2), 5 (1+2+2) and $3+4n$ for chain, pendulum and swimmer problems respectively, where n is the number of swimmer segments. In the LSPI algorithm, we set $\epsilon = 0.0001$ and limit the number of iterations to 20. The samples for each problem are collected by running a random policy, which uniformly selects one of possible actions, for a certain number of time steps (or episodes). In cascade correlation network, we trained an equal number of candidate units having Gaussian and sigmoid activation functions using RPROP method [18]. In RPROP, instead of directly relying on the magnitude of the gradient for the updates (which may lead to slow convergence or oscillations depending on the learning rate), each parameter is updated in the direction of the corresponding partial derivative with an individual time-varying value. The update values are determined using an adaptive process that depends on the change in the sign of the partial derivatives. We allowed at most 100 passes over the sample set during the training of candidate units, and employed the following parameters: $\triangle_{min} = 0.0001, \triangle_{ini} = 0.01, \triangle_{max} = 0.5, \eta^- = 0.5, \eta^+ = 1.2$. Although we opted for the regular RPROP algorithm, it is also possible to use improved and more robust versions [7].

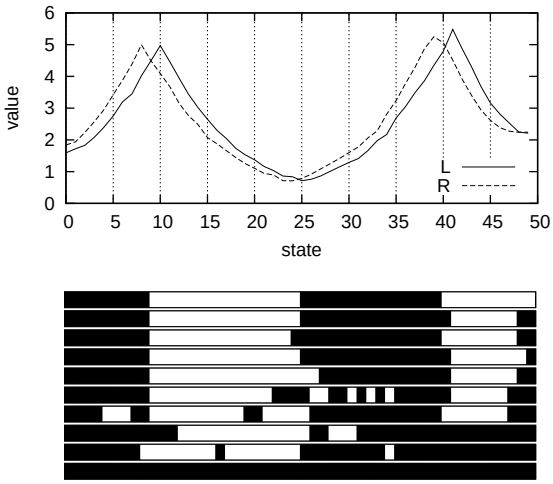


Figure 3. 50-state chain. (State-action value functions with 20 basis functions (top), and policy after every 2 basis functions (bottom).

Figure 3 shows the results for the 50-state (numbered from 0 to 49) chain problem using 5000 samples from a single trajectory. Reward is given only in states 9 and 40, therefore the optimal policy is to go right in states 0-8 and 25-40, and left in states 9-24 and 41-49. The number of candidate units was 4. In [10], using 10000 samples LSPI fails to converge to the optimal policy with polynomial basis function of degree 4 for each action, due to the limited representational capability, but succeeds with a radial basis function approximator having 22 basis functions. Using cascade-correlation basis expansion, after 10 basis functions near-optimal policies can be obtained and after 20 basis functions it converges to the optimal policy. The basis functions start from simpler ones and get more complex in order to better fit the target function.

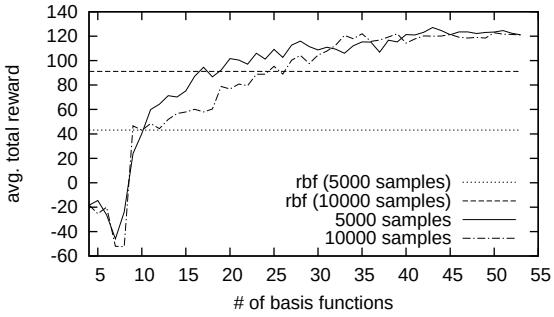


Figure 4. The progress of learned policies after each new basis function in the pendulum problem.

The results for the pendulum problem are presented in Figure 4. For this problem, we collected two different sets of samples of size 5000 and 10000, restarting from a random configuration every 20 time steps, and trained 10 candidate units. For both data sets, we also run LSPI using a radial basis function approximator on a 16×16 regular grid for each action. This corresponds to a set of 514 basis functions, including two bias terms. The performance of the resulting policies are evaluated by running 1000 episodes of 250 time steps each, and calculating the average of total reward. In the figure, we

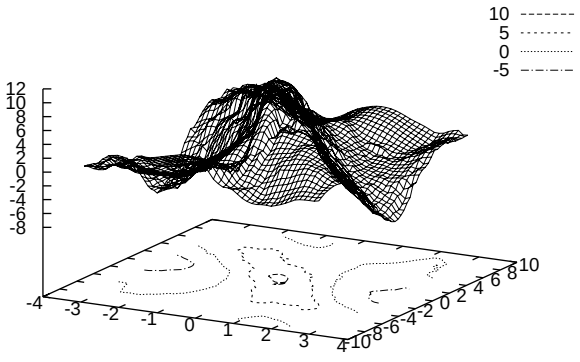


Figure 5. $\hat{Q}(s, \hat{\pi}(s))$ for the pendulum problem where $\hat{\pi}$ is the resulting policy. Although reward formulation is slightly different, see [3] for comparison.

present average over 30 such independent runs. As it can be clearly seen from Figure 4, with less number of samples the performance of policies obtained by radial basis function approximator decrease drastically. On the other hand, for both cases, the performance of the policies found by LSPI algorithm using the discovered basis functions converge to the same level. We also observe a consistent improvement as new basis functions are added (except in the very first iterations), yielding better performance levels using much less number of basis functions (10 and about 20 basis functions in case of 5000 and 10000 samples, respectively). Also, the value function obtained after 40 iterations is very close to the true one and successfully captures the shape of the function including the sharp edges around the ridge (Figure 5).

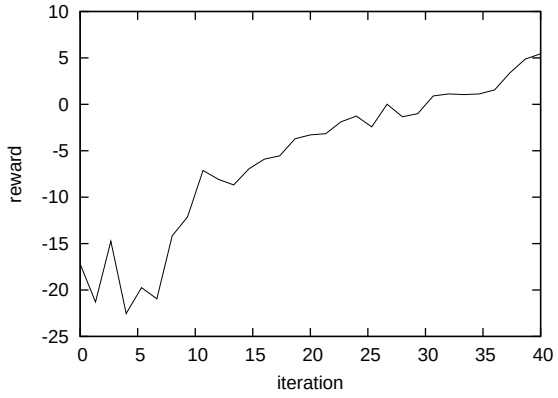


Figure 6. The results for the 5-segment swimmer.

We observed a similar behavior on more complex 5-segment swimmer problem as presented in Figure 6. For this problem, we collected 100000 samples restarting from a random configuration every 50 time steps. The number of trained candidate units was 10 as in the pendulum problem, but we allowed RPROP to make more passes (a maximum of 200) over the sample set. We again evaluated the policies by running 1000 episodes of 250 time steps each, and calculating the average of total reward.

6 Related Work

Basis function, or feature, selection and generation is essentially an information transformation problem; the input data is converted into another form that "better" describes the underlying concept and relationships, and "easier" to process by the agent. As such, it can be applied as a preprocessing step to a wide range of problems and have been in the interest of the data-mining community, in particular for classification tasks. Following the positive results obtained using efficient methods that rely on basis functions (mostly, using linear approximation architectures) in various domains, it also recently attracted attention from the RL community.

In [14], Menache et al. examined adapting the parameters of a fixed set of basis functions (i.e, center and width of Gaussian radial basis functions) for estimating the value function of a fixed policy. In particular, for a given set of basis function parameters, they used $LSTD(\lambda)$ to determine the weights of basis functions that approximate the value function of a fixed control policy, and then applied either a local gradient based approach or global cross-entropy method to tune the parameters of basis functions in order to minimize the Bellman approximation error in a batch manner. The results of experiments on a grid world problem show that cross-entropy based method performs better compared to the gradient based approach.

In [9], Keller et al. studied automatic basis function construction for value function approximation within the context of LSTD. Given a set of trajectories and starting from an initial approximation, they iteratively use neighborhood component analysis to find a mapping from the state space to a low-dimensional space based on the estimation of the Bellman error, and then by discretizing this space aggregate states and use the resulting aggregation matrix to derive additional basis functions. This tends to aggregate states that are close to each other with respect to the Bellman error, leading to a better approximation by incorporating the corresponding basis functions.

In [16], Parr et al. showed that for linear fixed point methods, iteratively adding basis functions such that each new basis function is the Bellman error of the value function represented by the current set of basis functions forms an orthonormal basis with guaranteed improvement in the quality of the approximation. However, this requires that all computations are exact, in other words, are made with respect to the precise representation of the underlying MDP. They also provide conditions for the approximate case, where progress can be ensured for basis functions that are sufficiently close to the exact ones. Their application in the approximate case on LSPI is closely related to our work, but differs in the sense that a new basis function for each action is added at each policy-evaluation phase by directly using locally weighted regression to approximate the Bellman error of the current solution.

In contrast to these approaches that make use of the approximation of the Bellman error, including ours, the work by Mahadevan et al. aims to find policy and reward function independent basis functions that captures the intrinsic domain structure that can be used to represent any value function [12, 8, 13]. Their approach originates from the idea of using manifolds to model the topology of the state space; a state space connectivity graph is built using the samples of state transitions, and then eigenvectors of the (directed) graph Laplacian with the smallest eigenvalues are used as basis functions. These eigenvectors possess the property of being the smoothest functions defined over the graph and also capture the nonlinearities in the domain, which makes them suitable for representing smooth value functions.

To the best of our knowledge, the use of cascade correlation networks in reinforcement learning has rarely been investigated before. One existing work that we would like to mention is by Rivest and Precup (2003), in which a cascade correlation network together with a lookup-table is used to approximate the value function in an on-line temporal difference learning setting [19]. It differs from our way of utilizing the cascade correlation learning architecture to build basis functions in the sense that in their case, cascade correlation network purely functions as a cache and an approximator of the value function, trained periodically at a slower scale using the state-value tuples stored in the lookup-table.

Finally, we are currently investigating various tracks in our own team. In particular, we would like to mention the use of genetic programming to build useful basis of features [6]. Using a genetic programming approach opens the possibility to obtain automatically human-understandable features. We also investigate a kernelized version of the LARS algorithm [11]. This basically selects the set of best features to represent a given function, according to set of sample datapoints; the features are automatically generated as in any kernel method. Furthermore, our approach is neither restricted to LSPI, nor

value-based reinforcement learning; [5] demonstrates that the same kind of approach may be embedded in natural actor-critics.

7 Conclusion

In this paper, we explored a new method that combines cascade correlation learning architecture with least-squares policy iteration algorithm to find a set of basis function that would lead to a better approximation of the state-action value function, and consequently results in policies with better performance. The experimental results indicate that it is effective in discovering such functions. An important property of the proposed method is that the basis function generation process requires little intervention and tuning from the user.

In the proposed method, LSPI is run to completion at each iteration, and then a new basis function is generated using the cascade correlation training (Algorithm 2). This benefits from a better approximation for the current set of basis functions. An alternative approach would be to add new basis functions within the LSPI loop after the policy evaluation step. This may lead to better intermediate value functions and steadier progress towards the optimal solution, but the resulting basis functions may not be useful at later iterations. It is also possible to combine both approaches by temporarily adding new basis functions within the LSPI loop and then discarding them. We are currently investigating these possibilities.

Although, our focus was on LSPI algorithm in this paper, the approach is in fact more general and can be applied to other reinforcement learning algorithms that approximate the state(-action) value function with a linear architecture. We pursue future work in this direction and also apply the method to more complex domains.

REFERENCES

[1] Dimitri Bertsekas and Sergey Ioffe, 'Temporal differences-based policy iteration and applications in neuro-dynamic programming', Technical Report LIDS-P-2349, MIT, (1996).
[2] Dimitri P. Bertsekas and John N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.
[3] Rémi Coulom, *Reinforcement Learning Using Neural Networks with Applications to Motor Control*, Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2002.
[4] Scott E. Fahlman and Christian Lebiere, 'The cascade-correlation learning architecture', in *Advances in Neural Information Processing Systems*, ed., D. S. Touretzky, volume 2, pp. 524–532, Denver 1989, (1990). Morgan Kaufmann, San Mateo.
[5] Sertan Girgin and Philippe Preux, 'Basis expansion in natural actor critic methods', in *Proceedings of the* 8^{th} *European Workshop on Reinforcement Learning*, (June 2008).
[6] Sertan Girgin and Philippe Preux, 'Feature discovery in reinforcement learning using genetic programming', in *Proceedings of Euro-GP*, volume 4971 of *LNCS*, pp. 218–229. Springer-Verlag, (March 2008).
[7] Christian Igel and Michael Husken, 'Empirical evaluation of the improved rprop learning algorithms', *Neurocomputing*, **50**, 105–123(19), (January 2003).
[8] Jeff Johns and Sridhar Mahadevan, 'Constructing basis functions from directed graphs for value function approximation', in *ICML '07: Proceedings of the 24th international conference on Machine learning*, pp. 385–392, New York, NY, USA, (2007). ACM.
[9] Philipp W. Keller, Shie Mannor, and Doina Precup, 'Automatic basis function construction for approximate dynamic programming and reinforcement learning', in *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pp. 449–456, New York, NY, USA, (2006). ACM.
[10] Michail G. Lagoudakis and Ronald Parr, 'Least-squares policy iteration', *Journal of Machine Learning Research*, **4**, 1107–1149, (2003).
[11] Manuel Loth, Manuel Davy, and Philippe Preux, 'Sparse temporal difference learning using LASSO', in *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, (April 2007).
[12] Sridhar Mahadevan, 'Representation policy iteration.', in *UAI*, pp. 372–379. AUAI Press, (2005).
[13] Sridhar Mahadevan and Mauro Maggioni, 'Proto-value functions: A laplacian framework for learning representation and control in markov decision processes', *Journal of Machine Learning Research*, **8**, 2169–2231, (2007).
[14] Ishai Menache, Shie Mannor, and Nahum Shimkin, 'Basis function adaptation in temporal difference reinforcement learning', *Annals of Operations Research*, **134**, 215–238(24), (February 2005).
[15] Rémi Munos, 'Error bounds for approximate policy iteration', in *ICML '03: Proceedings of the 20th international conference on Machine learning*, eds., Tom Fawcett and Nina Mishra, pp. 560–567. AAAI Press, (2003).
[16] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman, 'Analyzing feature generation for value-function approximation', in *ICML '07: Proceedings of the 24th international conference on Machine learning*, pp. 737–744, New York, NY, USA, (2007). ACM.
[17] Martin L. Puterman, *Markov Decision Processes — Discrete Stochastic Dynamic Programming*, Wiley, 1994.
[18] Martin Riedmiller and Heinrich Braun, 'A direct adaptive method for faster backpropagation learning: the rprop algorithm', pp. 586–591 vol.1, (1993).
[19] François Rivest and Doina Precup, 'Combining td-learning with cascade-correlation networks', in *ICML '03: Proceedings of the 20th international conference on Machine learning*, eds., Tom Fawcett and Nina Mishra, pp. 632–639. AAAI Press, (2003).
[20] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998. A Bradford Book.

1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2008)
Patras, Greece, July 22 2008

Application of Reinforcement Learning in a Real Environment Using an RBF Network

Sebastian Papierok[1] and **Anastasia Noglik**[2] and **Josef Pauli**[2]

Abstract. The application of reinforcement learning algorithms in the context of *robot behaviour learning* is a poorly explored and very promising area of research. In the present work the learned strategy which resulted from simulation has been applied in a real world environment. To achieve good results in the real world it was necessary to build a simulation environment which mirrors the reality up to a practicable degree. We use a radial basis function network to approximate the action-value function. To enforce a robot to learn a desired behavior a special online reward model has been developed. The approach reality-simulation-reality has been used to optimise the learning process in the simulation and apply the method in reality afterwards. Additionally the advantages and disadvantages of the application of the RBF-features over coarse coding with binary features have been examined.

1 Introduction

This article deals with a robot navigation problem in which an intelligent system should learn autonomously to navigate a mobile robot through a test track without collisions and in adequate time. For perception of the environment the robot's infrared sensors are used whose data are very noisy. A similar problem was considered in a simulation in [6]. Environment perception could also be done e.g. visual-based [1].

To realise reinforcement learning [9] the popular Sarsa(λ) approach will be applied in our studies. The action-value function will be approximated with an RBF network [7] [5], which will be used to control the robot's drive system. Because of the short training time and high accuracy of the RBF neural networks this method can be applied on real-time problems. The noisy sensor values will be converted here into a control-signal. This method doesn't need any information about the position and orientation of the infrared sensors so that a high level of flexibility can be ensured for the complete system.

The simulation should stick to the reality as closely as possible in order to make this method applicable on the real robot at a later time. For these purposes recorded real sensor data will be used in the simulation in order to use the learned control network in reality without the need for any further adaptation steps (see also [4]). This approach is described in [8] as virtual prototyping. An advantage of this approach is that the implemented algorithm can be tested under ideal conditions to determine appropriate learning parameters and clear potential faults as well. Afterwards, the complete learning process will take place in the real world environment.

[1] email: sebastian.papierok@stud.uni-due.de
[2] Universität Duisburg-Essen, Lehrstuhl Intelligente Systeme, Germany, email: {anastasia.noglik, josef.pauli}@uni-due.de

The applied fitness function for evolutionary algorithms from [2] was used as a basis for the development of an online reward model in this work. In [2] a control program for a similar problem at which a population-based EANT approach is applied was developed. There the EA approach was used to create a neural network that controls the robot's drive system. However, the downside of this method is that the learning process is very complex in reality. As an example: for a single episode 100 robots and 100 runs become necessary.

Practical experiments show that robustness and learning ability of the robot perform better by applying behaviour learning with RBF networks in comparison to coarse coding approximation. The application of an online reward model shows promising results.

2 Background of Reinforcement Learning

2.1 The Reinforcement Learning Problem

In the field of reinforcement learning an agent should learn autonomously to achieve specific goals. The agent observes the current state of the environment and makes action decisions depending on the current state. As a reaction for the executed actions the agent receives the next state and a numerical reward. By means of the reward the agent can evaluate its action decisions and improve its behavior over time. An appropriate reward model is needed in order to ensure that the agent achieves the given goals. The agent's objective is to maximize the sum of the rewards, also referred to as return, over the time. RL algorithms therefore try to estimate the expected return with the aid of so-called value functions.

2.2 Temporal Difference Learning

Temporal difference methods (TD methods) don't need a predefined model of the environment. They learn only by the experience that the agent gains by interacting with the environment. Moreover the agent can use its new experiences immediately to improve its behaviour. For estimation of the value function the states and rewards that the agent observes while interacting with the environment are used:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

By using continuous state spaces function approximators are used for estimating the value function based on samples that are observed during the interaction between agent and environment. Function approximators have a generalisation-ability so that the learning process occurs not only for one state-action pair but also for similar state-action pairs. The agent therefore must not visit all state-action pairs to make good action decisions. A function approximator can be considered as a function that is dependent on a real-valued parameter vector.

At TD methods the agent can use his new experiences immediately. Therefore, a method is needed that allows an incremental adaptation of the approximated value function. For this purpose function approximators in conjunction with gradient-descent methods are used.

2.3 Linear Approximation of the Value Function

The approximated value function depends linearly on the parameter vector of the function approximator. The relation between the parameter vector and the real states will be described by so-called features that are collected in a feature vector. The features are distributed in the real state space and have a determined dimension and shape. Usually they are reciprocally arranged so that each possible state will be described at least by one feature.

When applying coarse coding the features can hold only the values 0 and 1 and are hence described as binary features. A feature has the value 1 if it represents the current state or 0 if not.

Radial basis functions are a generalisation of coarse coding. The co-domain of this features is located in the intervall [0,1]. An RBF-feature i can be defined as a Gauss error distribution curve that has a given width σ_i and position c_i:

$$\phi_s(i) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$$

3 Concept for Combining Reinforcement-Learning with an RBF Network

3.1 RBF Network for Approximating the Value Function

A separate parameter vector $\vec{\theta}_{a,t}$ is used for each action a. This vector has the same number of components as the feature vector $\vec{\phi}_s$. The approximated action-value function is defined as:

$$Q_t(s,a) = \vec{\theta}_{a,t}^T \vec{\phi}_s = \sum_{i=1}^{n} \theta_{a,t}(i)\phi_s(i)$$

Figure 1 shows the RBF network that is used to approximate the action-value function.

3.2 Gradient-Descent Sarsa(λ)

Figure 2 shows the gradient-descent Sarsa(λ) algorithm. The base algorithm of [9] was modified so that radial basis functions are used as features. Moreover it was adapted to the structure of the RBF network of figure 1.

The algorithm terminates if the specified number of episodes has been reached. An episode ends if the maximum number of steps has been reached or the agent has found the specified goal. At a collision of the robot with an obstacle the actual episode will be finished as well because the robot is only allowed to move forward. The problem is that the pivot point of the robot is located slightly in front so that a rotation is not sufficient to rescue it from this situation.

In the base algorithm of [9] the set $\mathcal{F}$ contains the indices of the features that represent the current state-action pair (s, a). In this work the set $\mathcal{F}_s$ is only dependent on the current state s. The action a is required once in the output layer of the RBF network. This approach has the advantage that the number of features is independent of the number of actions. Moreover the calculation effort can be reduced this way if the Q-value for only one state-action pair is required, which is the case when the agent chooses a random action.

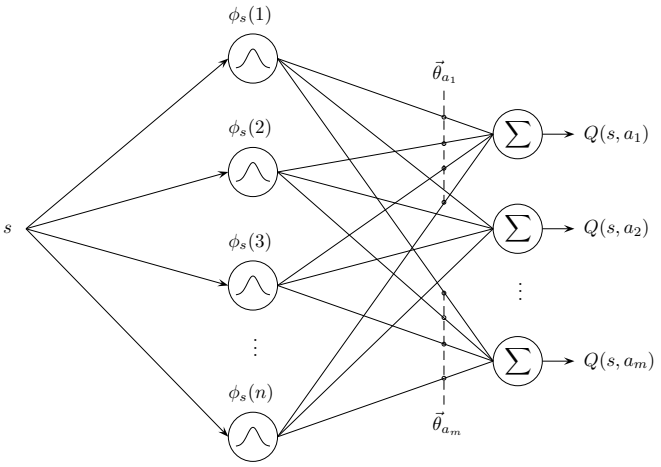


Figure 1. RBF network for the approximation of the action-value function.

```
01 Initialize θ⃗ abitrarily
02 Repeat (for each episode):
03        e⃗ = 0⃗
04        s, a ← initial state and action of episode
05        F_s ← set of features present in s
06        Repeat (for each step of episode):
07              For all i ∈ F_s:
08                    e⃗_a(i) ← e⃗_a(i) + φ_s(i)
09              Take action a, observe reward, r,
                      and next state, s'
10              δ ← r − Σ_{i∈F_s} θ_a(i)φ_s(i)
11              With probability 1 − ε:
12                    For all a ∈ A(s'):
13                          F_{s'} ← set of features
                                  present in s'
14                          Q_a ← Σ_{i∈F_{s'}} θ_a(i)φ_{s'}(i)
15                    a' ← arg max_a Q_a
16              else
17                    a' ← a random action ∈ A(s)
18                    F_{s'} ← set of features present in s'
19                    Q_{a'} ← Σ_{i∈F_{s'}} θ_{a'}(i)φ_{s'}(i)
20              δ ← δ + γQ_{a'}
21              θ⃗ ← θ⃗ + αδe⃗
22              e⃗ ← γλe⃗
23              s ← s'
24              a ← a'
25        until s is terminal
```

Figure 2. Linear, gradient-descent Sarsa(λ) with RBF-features and ϵ-greedy policy.

3.3 State Space

For perception of the environment the infrared sensors of the robot are used. The state space is defined as $([10, 65] \cup \{-1\})^7$. The interval $[10, 65]$ specifies the used measurement range of the infrared sensors. The value -1 is used if the sensors does not detect any obstacle. A state is composed of the measured values of seven infrared sensors. There are three RBF features per dimension of the state space so that the 7-dimensional state space has a total of 3^7 features. Figure 3 shows the distribution of the features for one sensor.

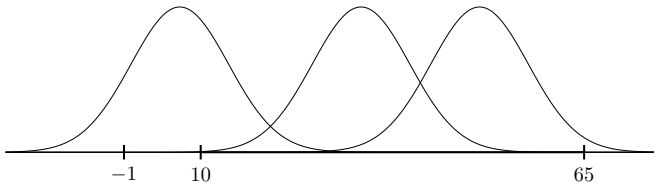


Figure 3. Distribution of the radial basis functions for one infrared sensor.

Again, the sensor values are very noisy. At an ideal distance of 45 cm the sensors supply values between 35 cm and 70 cm [3]. Moreover there are sensor failures at which the obstacles will not be detected at all. Figure 4 shows the simulated infrared sensors of the robot.

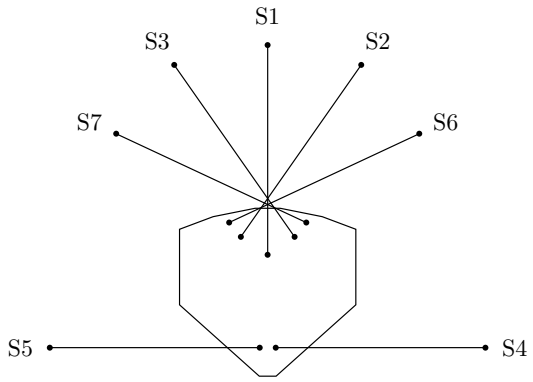


Figure 4. Simulated infrared sensors of the robot.

3.4 Action Set

The action set consists of three actions: turn right, move forward, and turn left. An action is defined as a tuple $(\Delta\varphi, \Delta s)$ at which $\Delta\varphi$ describes an angle of rotation and Δs a stroke for moving forward. It is assumed that the absolute orientation and position of the robot are unknown so that an action describes only the change of orientation and position respectively. The forward and rotational velocity are assumed as constant. For the reward model the actions are defined as:

$$a \in \begin{cases} -1 & \text{turn right} \\ 0 & \text{move forward} \\ 1 & \text{turn left} \end{cases}$$

4 Online Reward Model

The intention at the development of a reward model is to provide the robot a specific behaviour in specific situations under the additional condition that the number of episodes should be as low as possible. A similar problem was treated in [4] (by our working group) with evolutionary algorithms (EA) and was successfully used in reality.

In the following the similarities between the evolutionary approach and the reinforcement learning approach will be discussed. Candidate solutions to the optimisation problem play the role of individuals in a population (for example in [2]) where an individual is the artificial neural network that controls the robot. The fitness function determines the learning process of the individual. Individuals in EA are with Q^{π}_{app} defined policies (here the RBF networks) that are comparable with the RL approach. The fitness function of the individuals is almost comparable with the approximation of the value function Q^{π}_{app} for policy π.

Because of this similarity the elementary reward of the developed reward model is based on the fitness function from [2]. That fitness function favors the desired behaviour of the robot as well. The fitness function is defined as $F = \sum_{t=1}^{T} f(t)$, where $f(t) = v(t)exp(-100(H(t) - H(t-1))^2)s^{min}(t)$ is the according value at time t. $v(t)$, $H(t)$ and $s^{min}(t)$ are the speed, the heading of the robot, and the minimal value of the sensor set readings respectively. This fitness function favors controllers that move straight as long and as fast as possible and controllers that navigate the robot with the maximum distance from the obstacles.

Observations have shown that the penalty mechanism at the evolutionary approach is built in but not easy to find. This is relevant, if the individual stands still due to a collision and will not be rewarded further. The population orientated method through several possible solutions is leading quickly to the desired solution.

In the reward model that is used in this work the penalty mechanism will be introduced artificially. The minimal sensor value of the sensor set is defined as $s^{min}_t := min_i s_t(i)$. The penalty mechanism will be activated if the minimal sensor value falls below a critical threshold s^{min}. The value s^{min} depends on the design of the robot, its balance point, and its dimensions. With the help of the $a^{desired}$ value a kind of reflex will be introduced that acts rational. In case the robot receives a message from the right side about a shortfall of the minimal acceptable distance s^{min} the rationality says: turn right and move forward is not the correct decision so that it applies to $a^{desired}_t = 1$. The negative elementary reward will be computed depending on $(s^{min} - s^{min}_t)$ and the value $|a^{desired}_t - a_t|$. The complete reward model is defined as

$$r_t(s^{min}_t, a_t, a_{t-1}) = \begin{cases} -1 \cdot (s^{min} - s^{min}_t)^{|a^{desired}_t - a_t|+1} & : \text{ for negative reward} \\ (s^{min}_t - s^{min}) \cdot \exp\left(-\frac{(a_t - a_{t-1})^2}{2\sigma 2}\right) & : \text{ for positive reward} \end{cases}$$

where $\sigma = 0.75$. The elementary reward is hence dependent on the current state s_t as well as the two last executed actions a_t and a_{t-1}. It concerns a reflex that protects the agent's life. It is important to find a balance between rewards and penalties. The reward for goal reaching should at least differ by a factor of 10.

5 Results and Discussion

There were several tests performed to determine the robustness and the learning ability of the RBF network which controls the robot's

drive system. The evaluations have shown that the used RBF network has the ability to process the sensor data so that interferences like sensor failures or noisy sensor values have no strong influences on the learning process and the application of the method. It could be shown that the simulated robot can navigate through the given test track after few episodes (after one episode as well) without collisions. The knowledge that was learned in the simulation was applied in the real world without additional effort so that the robot was able to navigate through the real test track without collisions as well.

5.1 Comparison between RBF and Coarse Coding

In the following the characteristics of the intelligent system such as robustness and efficiency will be determined by applying of radial basis functions and coarse coding.

The robustness and efficiency of the method will be considered in the *Map World*. This is defined as an extension of the *Grid World* [9] to a continuous state space which is represented by the position and orientation of the robot. Feedback from the environment in *Map World* is the robot's position and orientation in the world coordinate system. The start and goal positions are fixed. This environment was chosen to better understand the learning process of the implemented algorithm. The behaviour of the robot is statistically not easy to interpret. In contrast, the number of steps to a fixed goal is always ascertainable. Consequently, the statements about the robustness and efficiency of the learning process can be derived easier. Moreover, the required time to solve the task in the *Map World* is much less compared to the measuring space. As a result the number of trials and the significance of the results can be increased. It is assumed that the characteristics of the method are portable to other state spaces as well.

5.1.1 Test Method

To compare RBF and coarse coding the position and orientation of the robot are used for perception of the environment so that the agent optimises the path from the start position to the specified goal. The agent receives a positive reward if it finds the goal. In the case of a collision of the robot with an obstacle the agent gets a penalty (negative reward). In the other cases it receive a small penalty. Figure 5 shows the used simulated environment of the robot. The analysis showed that the differences between RBF and coarse coding are significant even in the case of a very easy test environment.

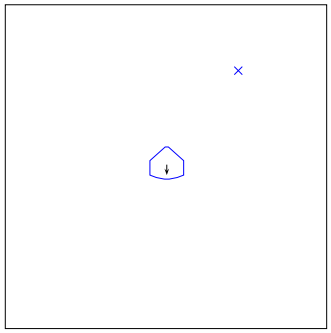


Figure 5. Start configuration of the test environment (environment 1). The arrow describes the robot's driving direction.

The learning rate α will be set to the value $0.2/|\mathcal{F}_s|$ during the learning process at which $|\mathcal{F}_s|$ describes the number of features that represent the current state [9]. The discount factor γ and the decay factor λ are both set to the value 0.8. The number of episodes is set to the value 400 for environment 1 at which one episode consists of maximal 200 steps. A total of 200 trials are performed for each test.

To understand the learning process an additional episode will be passed between each second episode at which the learning rate and the exploration rate are both set to the value 0. It will be described hereafter as sample. A sample describes the number of steps that the simulated robot needs from the start position to the given goal. After the additional episode the learning process will be continued.

The following diagrams show the average values of the samples from 200 trials at which the simulated robot has reached the goal. To compute the number of samples that were used for the computation of a specific average value the probabilities for goal reaching are shown in the diagrams as well.

5.1.2 Different Exploration Rates

In the following the convergence properties of the method using different exploration rates ϵ will be analysed.

At a sensor failure the current state will be described by another subspace of the state space. This means that the agent can choose an action that is possibly inapplicable in the present situation. At a large measurement difference of a sensor the agent can choose a wrong action as well because the current state will be possibly represented by other features. Such behaviour of the agent is comparable with the selection of random actions.

Figure 6 shows the results for $\epsilon = 0.05$ using environment 1. The method provides good results for both coarse coding and radial basis functions. The optimal path was found after 130 episodes for both types of features.

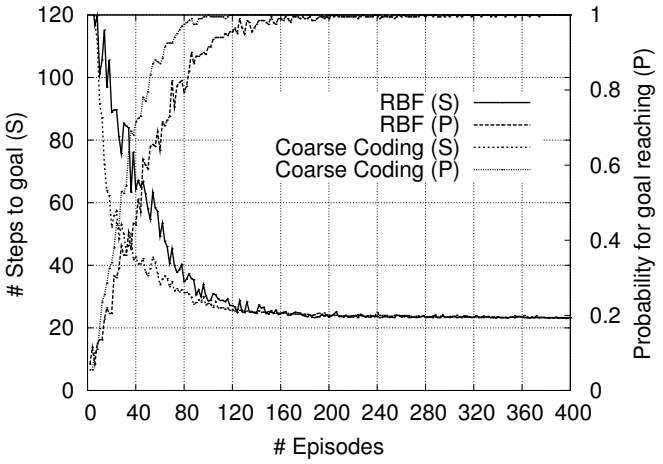


Figure 6. Comparison between RBF and coarse coding using a small exploration rate ϵ. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.05$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 7$.

Figure 7 shows the results for $\epsilon = 0.6$. It is noticeable that the method is susceptible to interferences by using coarse coding while it is stable and converges towards a specific value after 240 episodes using radial basis functions. It is noticeable that the probability for goal reaching by using coarse coding no longer achieves the value

1 but takes course highly unstable at smaller values. Moreover the optimised path by using radial basis functions is a bit better than in figure 6.

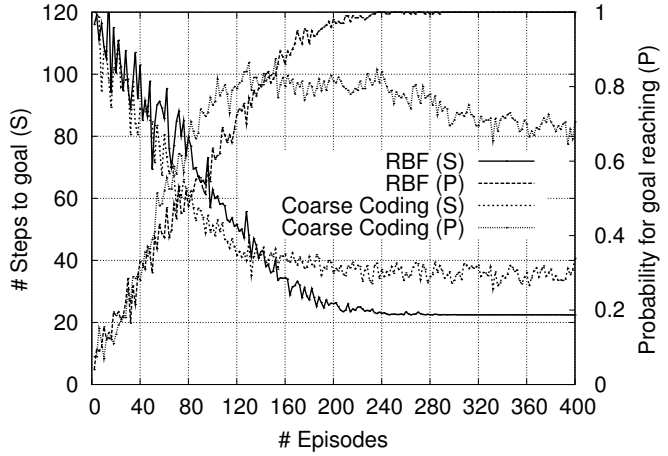


Figure 7. Comparison between RBF and coarse coding using a high exploration rate ϵ. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.6$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 7$.

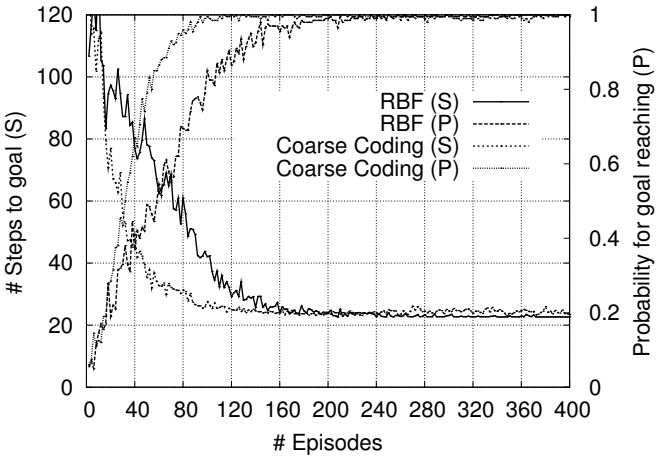


Figure 8. Comparison between RBF and coarse coding using a small number of feature overlaps. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.2$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 7$.

Binary features also have heavy effects if the current state is located far away from the centre of a feature so that the computed linear combination contains components of the parameter vector that are not important for the representation of the current state as well. At small exploration rates state-action pairs already known by the agent will be visited again and again so that the agent does not differ from its current strategy. At high exploration rates the agent visits unknown state-action pairs time after time. It is believed that the agent forgets its gathered knowledge if it reaches a state-action pair at which it must discover the goal again. This effect will be enforced as a result of long eligibility traces because the before attended state-action pairs will be adapted over a longer period of time.

The described effect can not occur when using radial basis functions because the components of the parameter vector are included only scaled in the computation of the approximated action-value function.

5.1.3 Different Dimensions of the Features

In the following several tests are performed to determine the robustness of the method when using different dimensions of the features. For this reason the width of the features will be modified i.e. support of Gaussian. It is assumed that the features are distributed in the state space evenly so that each state can be described at least by one feature. To analyse the results independently of the position and dimension of the features the minimum and maximum number of overlaps will be determined. The exploration rate will be set to the value 0.2 to simulate a small interference.

Figure 8 shows the results for $\sigma = 7$. It can be noticed that the method provides a good result for both coarse coding and radial basis functions.

The results for $\sigma = 11$ are shown in figure 9. By using coarse coding the same effect can be observed here as in figure 7.

By using coarse coding the difference $Q_{CC}(s,a) - Q_{RBF}(s,a)$ will increase with the growing number of overlaps because the com-

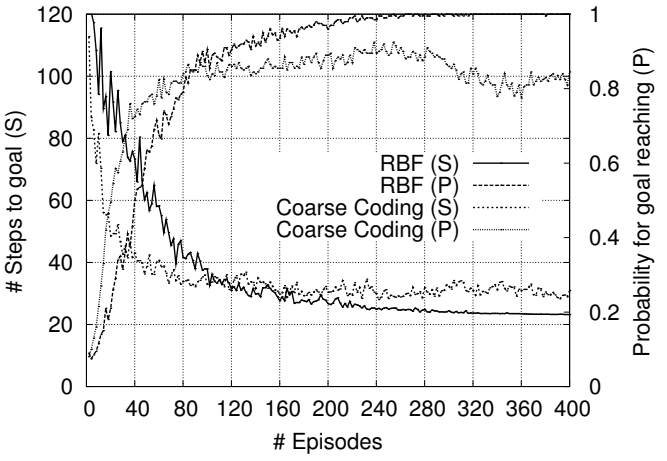


Figure 9. Comparison between RBF and coarse coding using a high number of feature overlaps. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.2$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 11$.

puted linear combinations have more and more components of the parameter vector that are not very important for the representation of the current state. This means that the number of features is growing in which the current state is located far from the centre.

By using RBF features the method converges towards a specific value, but the number of required episodes is higher than in figure 8. This is because the adaptation of the parameter vector requires more time in conjunction with a growing number of overlaps, because the sensibility of the computed linear combination is growing already at small state changes. Therefore, a high number of overlaps has negative effects on the method.

The method provides good results if each possible state is represented at least by two features. The number of overlaps must consequently not be that high in order to use this method efficiently and robustly.

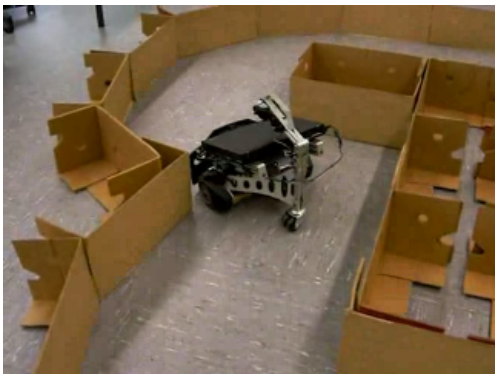


Figure 10. The robot in the real world.

5.2 Learning of Behaviour in Online Mode

As stated before, the agent should learn to navigate the robot through a given test track without collisions and in adequate time. In the first instance the process pattern reality-simulation-reality was used. In this case, the learning process took place in the simulation.

Several reward models have been tested in the simulation. The intelligent system uses an online reward model with which the agent can improve his behaviour while it is navigating the robot through the test track. The agent gathers its knowledge because of the states, actions, and rewards that occur during the interaction with the environment. There is a test track given at which the agent has to learn the correct behaviour without previous knowledge.

Observations have shown that the robot was already able to navigate through the test track after a few episodes. A collision can happen once in a while if several sensor failures or wrong sensor values respectively occur in a row. To reduce the influences of interferences the forward velocity of the robot can be decreased. Afterwards the knowledge that was learned in the simulation was applied on the real robot. In reality the robot could navigate through the test track as well.

The learning rate α is set to the value 0.2 during the learning process. The exploration rate ϵ is adjusted to the value 0.01 in order that the agent shall generally use its gathered knowledge to navigate the robot through the test track. The discount factor γ and the decay factor λ are both set to the value 0.8.

In the second instance the complete learning process has been performed in reality (see figure 10). The agent has to learn the correct behaviour without previous knowledge here. Observations have shown that few episodes are needed in order that the agent can navigate the robot through the test track without collisions. The learning process in the reality has been recorded on video[3].

The intelligent system has the ability of generalisation so that the agent can learn its behaviour in a complex test track and use it in yet unknown test tracks.

6 Conclusion

The developed method offers two advantages. It is stable and leads quickly to the desired behaviour of the robot in reality. The used approach is the Sarsa(λ) algorithm from the field of reinforcement learning. Because of the short learning time and the adaptation ability RBF-features have been used for approximation of the action-value function. The RBF network creates a control signal from the measured sensor values so that the desired behaviour of the robot can be determined. It was necessary to develop an online reward model. The fitness function of evolutionary algorithms was applied as the base of the online reward model. The adaptation ability of the RBF network was confirmed by comparing it with coarse coding in the benchmark environment *Map World*. For the complete method the virtual prototyping approach was used. Numerous trials have shown that the approach is fast, efficient, and adaptive on difficult real environments as well.

[3] see `http://www.uni-due.de/is/projekt_emrobnav.php`

REFERENCES

[1] D. Busquets, R. L. de Mo'ntaras, C. Sierra, and T. G. Dietterich, 'Reinforcement learning for landmark-based robot navigation', in *First International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 841–843, Bologna, (2002).

[2] Y. Kassahun, *Towards a Unified Approach to Learning and Adaptation*, Ph.D. dissertation, Christian-Albrechts-Universität, Institut für Informatik und Praktische Mathematik, Kiel, 2006.

[3] T. Köpsel, *Evolutionäre Algorithmen zur Topologieentwicklung von Neuronalen Netzen für die Roboter-Navigation im praktischen Einsatz*, Diploma Thesis, Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Duisburg, 2007.

[4] T. Köpsel, A. Noglik, and J. Pauli, 'Evolutionäre Algorithmen zur Topologieentwicklung von Neuronalen Netzen für die Roboter-Navigation im praktischen Einsatz', in *Autonome Mobile Systeme 2007*, ed., T. Luksch K. Berns, Informatik aktuell, pp. 145–151, Berlin, (2007). Springer-Verlag.

[5] J. Li and T. Duckett, 'Robot behaviour learning with a dynamically adaptive rbf network: experiments in offline and online learning', *CIRAS 2003, Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, (2003).

[6] K. Maček, I. Petrović, and N. Perić, 'A reinforcement learning approach to obstacle avoidance of mobile robots', in *Proceedings of the 7th IEEE International Workshop on Advanced Motion Control*, pp. 462–466, Maribor, (2002). IEEE.

[7] K. Samejima and T. Omori, 'Adaptive internal state space construction method for reinforcement learning of a real-world agent', *Neural Netw.*, **12**(7-8), 1143–1155, (1999).

[8] Richard Stansbury, Eric Akers, Hans Harmon, and Arvin Agah, 'Simulation and testbeds of autonomous robots in harsh environments', in *Software Engineering for Experimental Robotics*, ed., Davide Brugali, volume 30 of *Springer Tracts in Advanced Robotics*, Springer - Verlag, Berlin / Heidelberg, (April 2007).

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

1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2008)
Patras, Greece, July 22 2008

Using cooperative multi-agent Q-learning to achieve action space decomposition within single robots

Sebastiaan Troost[1] and **Erik Schuitema**[2] and **Pieter Jonker**[1,2]

Abstract. Reinforcement Learning (RL) is a promising way of controlling autonomous robotic systems. It is well known that RL does not scale very well towards systems with many inputs and outputs, like humanoid robots. Making RL more scalable towards more complex systems has become one of the most important issues in this research field. When considering a RL system with many outputs, we not only face the problem of storing the action-value function for all output combinations, we also need to evaluate all output combinations at every action selection, which quickly becomes computationally heavy. This paper tries to tackle the problems that come with large action spaces by using a multi-agent approach in which each output is controlled by an independent Q(λ)-learning agent. All agents receive the same rewards, i.e., they work in a fully cooperative setting. We test our approach on two simulated robotic systems, each with two actuators; a two-link manipulator and a bipedal walking robot. Both systems show that the multi-agent approach with two agents learns with a speed and system performance almost identical to the single-agent approach, while the memory requirements and action selection computation time is now linear in the number of actuators, instead of exponential. We also tested three modifications to our approach: using SARSA, using synchronized exploration and using Lenient Learning. With the two-link manipulator, Lenient Learning significantly increased learning speed and performance, while the other modifications didn't have a significant effect in either of both systems.

1 INTRODUCTION

Reinforcement Learning (RL) receives increasing attention as a way of controlling autonomous systems, mostly because of its mild assumptions on the learning problem and its capabilities to learn online. However, it is well known that RL does not scale well towards more complex systems, like humanoid robots, because of the large number of inputs (sensors) and outputs (actuators) that are spanning the continuous state-action space of these systems. The larger the state-action space of a system, the longer it takes and the more memory it requires to find a control policy with RL. While the state-action space as a whole has these two effects when getting larger, a large action space possesses an additional computational disadvantage. Suppose a robot has M motors, each of which can be controlled in N discrete actuation steps. Selecting the best action in a certain state involves evaluating N^M different state-action values and this has to be done at each time step. One way to overcome this large choice of actions is to work with parameterized policies for (groups of) actuators, like in policy gradient approaches [8, 22, 16]. While this approach significantly reduces the search space of the learning problem, it requires prior knowledge on sensible policy functions. Although this prior knowledge could be quite intuitive for certain robotic systems, it is, in our opinion, a step away from fully autonomous robots using a generic learning framework and can rule out inventive solutions that a programmer did not consider when defining the control policy functions.

In this paper, we choose to *decompose* the action space by assigning a learning agent to each individual actuator and letting those heterogeneous agents learn to achieve the overall system goals by cooperating. Each agent's state space consists of the full state space, but does not include any information on the action selection on the other actuators. In [4] this approach is called Independent Learners (IL).

In the IL approach, the state transitions for each agent do not only depend on the actions taken by that agent itself, but also on the actions of the other agents. Since these are not included in the agent's state-action space, the state transition function for each independent agent is in effect time-varying. Only when all policies have become static, for example when they all converged to the optimal ones, each agent's transition function is static and the problem for each agent is again a Markov Decision Process (MDP) [21]. The single-agent MDP is extended to a multi-agent MDP in [10, 2], however, no convergence proofs are found under the same conditions as for RL on a single-agent MDP [4]. By giving each agent the same reward for their joint action, thus only when *all* agents do the right thing simultaneously, this approach might still lead to convergence to an (almost) optimal solution of the whole system. When using this approach, instead of evaluating the action-values of all possible action combinations of the actuators, as in the single-agent case, each agent selects its own action, without any form of negotiation. In our example of M motors with N discrete actions, this will lead to evaluating and storing $M \cdot N$ values instead of N^M; an increase that is linear in the number of actuators instead of exponential.

We tested this approach on two different simulations of robotic setups: a two-link manipulator and a, more complicated, bipedal walking robot. Both setups have two motors that can be controlled to generate a certain torque. In both tests, classical single-agent Q(λ)-learning is compared with our multi-agent Q(λ)-learning approach. We also tested three modifications to our approach: using SARSA [21], an on policy learning algorithm, using synchronized exploration, where all agents explore at the same time, and using Lenient Learning [14], a method for multiple agents to be lenient to each other's actions. For the continuous state space, we use tile coding

[1] Delft University of Technology, Faculty of Applied Sciences, Dept. of Imaging Science and Technology, Lorentzweg 1, NL-2628CJ, Delft, The Netherlands
[2] Delft University of Technology, Faculty of 3ME, Dept. of Biomechanical Engineering, Mekelweg 2, NL-2628CD, Delft, The Netherlands, email: E.Schuitema@tudelft.nl, P.P.Jonker@tudelft.nl

[21] as a function approximator.

This paper is organized as follows. In Section 2, we explain the theory on RL, multi-agent RL (MARL) and our proposed method. In Section 3, we explain our first test setup consisting of a simulated two-link manipulator with two actuators. In Section 4, we explain our second test setup consisting of a simulated bipedal walking robot with two actuators. We finally present a discussion in Section 5 and our conclusions in Section 6.

2 THEORY

Reinforcement Learning (RL) is designed to find optimal control policies for Markov Decision Processes (MDPs) [21]. If a problem can be formulated as an MDP for discrete state-action spaces, two well-known temporal difference (TD) algorithms, Q-learning [24, 21] and SARSA [19, 21], are proven to converge to the optimal control policy for the MDP[3]. For our proposed approach, the MDP framework needs to be extended to a multi-agent version in which several learning entities are active in the same environment.

2.1 MMDP

A Markov Decision Process or MDP is defined as the 4-tuple:

$$\langle S, A, T, R \rangle, \tag{1}$$

where S and A are finite sets of the states and actions, where $T : S \times A \times S \longrightarrow [0, 1]$ is a transition function defining the probability of transitioning to state $s_{t+1} \in S$ when executing action $a_t \in A$ in state $s_t \in S$, and where $R : S \times A \rightarrow \Re$ is a real valued reward function. An MDP has the Markov-property, which means that the probability distribution function over the next states of the system *only* depends on the current state-action pair.
In [10, 2] this single-agent MDP is extended to a multi-agent version. This is done by including all agents in the tuple. We now define the multi-agent MDP (MMDP) as the 5-tuple:

$$\langle S, M, \{A^1, ..., A^n\}, T, R \rangle, \tag{2}$$

where M is a finite set of agents, A^i is the action space available to agent i, $T : S \times A^1 \times \ldots \times A^n \times S \longrightarrow [0, 1]$ is a transition function and $R : S \rightarrow \Re$ is a real valued reward function.

We propose to split this MMDP up by creating, for each agent M^i, a 4-tuple with a single action space A^i, the entire set of states S, a time-varying transition function only dependent on the agent specific actions A^i $T_{i,t} : S \times A^i \times S \rightarrow [0, 1]$ and the reward function R, based on the joint action $A = \{A^1, \ldots, A^n\}$:

$$M^i : \langle S, A^i, T_{i,t}, R \rangle. \tag{3}$$

The transition functions $T_{i,t}$ are not constant because the state S_{t+1} the system arrives in after taking action $a_i \in A^i$ depends on the joint action A. These 4-tuples, thus, do not possess the Markov-property unless $T_{i,t}$ are static distributions. This should happen when the policies of the agents become stationary, i.e., they converge.

[3] The convergence of Q-learning and SARSA combined with a function approximator to handle continuous state spaces is still a topic under research [23, 1, 6, 17].

2.2 MA Q(λ)-learning

Q-learning is an off-policy learning method, which means that the optimal policy is learned while action selection during learning may follow a different policy. With standard Q-learning, the total expected sum of rewards of choosing action a in state s and following the estimated *optimal* policy afterwards is estimated and stored as the Q-value of ($s_t \in S, a_t \in A$). It is updated after the agent executed a_t in s_t and observed the result:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta, \tag{4}$$

where $\alpha \in [0, 1]$ is the learning rate. For δ in (4) the Q-learning specific δ_Q is

$$\delta_Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t). \tag{5}$$

where $\gamma \in [0, 1]$ is the time discount factor. To increase convergence speed, Q-learning can be combined with replacing eligibility traces to form Q(λ)-learning [21]. This uses a trace to update all action values (Q) previously visited in the current episode, at every time t. Each visited location (s, a) in the action-value space is stored in this trace. At time t, all values in the trace are updated with the use of an intermediate function $e_t(s, a)$:

$$\begin{array}{l} e_t(s,a) = \left\{ \begin{array}{rl} 1, & s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a), & \text{otherwise} \end{array} \right. \\ Q_{t+1}(s,a) = Q_t(s,a) + \alpha e_t(s,a)\delta. \end{array} \tag{6}$$

e_t is initialized at 0 and set to 1 when $s = s_t$ and $a = a_t$. After each update, the factor $e_t(s, a)$ is decayed by $\gamma\lambda$, where $\lambda \in [0, 1]$ is the trace discount factor. When the action selection policy (see Section 2.4) chooses a different action a_{t+1} than $\arg\max_{a'} Q(s_{t+1}, a')$, the trace needs to be cleared because the state-action pairs in the trace can no longer all be held responsible for the current state under the optimal policy.

The update rule can be extended to a multi-agent Q(λ)-learning update rule to try and solve the previously proposed MMDP. We propose to give each Independently Learning (IL) [4] agent i its own multi-state action-value function Q^i spanned by $S \times A^i$ that is updated (note that general formal convergence guarantees are lost [4]) with a similar rule to (6):

$$\begin{array}{c} Q^1_{t+1}(s, a^1) = Q^1_t(s, a^1) + \alpha e^1_t(s, a^1)\delta^1, \\ Q^2_{t+1}(s, a^2) = Q^2_t(s, a^2) + \alpha e^2_t(s, a^2)\delta^2, \\ \vdots \\ Q^1_{t+1}(s, a^n) = Q^1_t(s, a^n) + \alpha e^n_t(s, a^n)\delta^n, \end{array} \tag{7}$$

where $e^i_t(s, a^i)$ is the agent i specific trace decay function:

$$e^i_t(s, a^i) = \left\{ \begin{array}{rl} 1, & s = s_t \text{ and } a^i = a^i_t \\ \gamma\lambda e^i_{t-1}(s, a^i), & \text{otherwise} \end{array} \right. \tag{8}$$

that for Q(λ)-learning is cleared when agent i performs an exploratory action, and δ^i is the agent specific δ^i_Q of (5) for MA Q(λ)-learning:

$$\delta^i_Q = r_{t+1} + \gamma \max_{a'^i} Q(s_{t+1}, a'^i) - Q(s_t, a^i_t). \tag{9}$$

2.3 MA SARSA(λ)-learning

SARSA is similar to Q-learning but is an on-policy algorithm. This means, that SARSA will learn to predict the total expected sum of

rewards of choosing action a in state s and following the *current* policy afterwards. The update rule (4) still applies however δ_S is now used for δ:

$$\delta_S = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t), \tag{10}$$

where a_{t+1} is the action chosen by the action selection policy. Extending SARSA to SARSA(λ) is the same as for Q-learning, with one exception: the trace does not need to be cleared when choosing an action a_{t+1} in the learning process that is not equal to $\underset{a'}{\text{argmax}} Q(s_{t+1}, a')$.

We again propose to extend SARSA(λ) to the multi-agent version MA SARSA(λ), which results in (7) with trace decay function (8), which for SARSA is not cleared when agent i performs an exploratory action. For δ^i the SARSA specific δ^i_S (10) is used:

$$\delta^i_S = r_{t+1} + \gamma Q(s_{t+1}, a^i_{t+1}) - Q(s_t, a^i_t). \tag{11}$$

2.4 Policy

Several action selection policies can be used during learning to incorporate different exploration strategies. We used the ϵ-greedy policy with the exploration rate ϵ:

$$a_t = \begin{array}{c} x = \text{random}([0,1]), \\ \begin{cases} \text{random}(a \in A), & x < \epsilon \\ \underset{a'}{\text{argmax}} Q(s_t, a'). & x \geq \epsilon \end{cases} \end{array} \tag{12}$$

In the case of multi-agent Q(λ)-learning each agent i has its own policy and thus independently decides which action a^i to take:

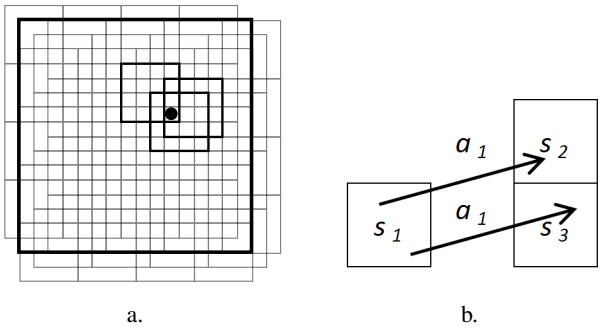
$$a^i_t = \begin{array}{c} x^i = \text{random}([0,1]), \\ \begin{cases} \text{random}(a^i \in A^i), & x^i < \epsilon \\ \underset{a'^i}{\text{argmax}} Q^i(s_t, a'^i). & x^i \geq \epsilon \end{cases} \end{array} \tag{13}$$

One may choose to synchronize exploration between the agents by always exploring simultaneously:

$$a^i_t = \begin{array}{c} x = \text{random}([0,1]), \\ \begin{cases} \text{random}(a^i \in A^i), & x < \epsilon \\ \underset{a'^i}{\text{argmax}} Q^i(s_t, a'^i). & x \geq \epsilon \end{cases} \end{array} \tag{14}$$

2.5 Function approximation: tile coding

We use tile coding [21] as a function approximator for continuous state-action spaces. With tile coding, first the state-action space is discretized into a multidimensional grid, a tiling, by defining the grid spacing, or tile width, in each dimension. Setting the resolution for each dimension is done using intuition and trial-and-error. Additionally, a number of such tilings are superimposed on this base tiling, evenly distributed within one tile width, see Figure 1a. The Q-value of a certain state-action pair is then approximated by averaging all values of the tiles that the state-action pair falls into. Because of the discretization of a continuous problem, the MDP at hand becomes a stochastic MDP; see Figure 1b. From state s_1 the same action can result in different states (s_2 and s_3), because the starting point, while within one discrete state, in the continuous world is slightly different.



a. b.

Figure 1. a. Illustration of tile coding for 3 tilings in 2 dimensions. The value at the dot is being evaluated. The values of the three tiles at the dot are averaged to produce the final function value. b. Illustration of taking action a_1 in state s_1. This action can transfer the system to state s_2 or s_3.

2.6 Action space decomposition

To make Reinforcement Learning more scalable in the number of actuators and more suitable for (humanoid) robots we propose to decompose the action space of the system. We do this by implementing multiple agents, each using Q(λ)-learning and each having control over one of the actuators. Its advantages can be easily pictured by Table 1, where the action space is shown for two actuators A and B.

Table 1. Action space of a. single-agent Q-learning b. multi-agent Q-learning

a.

	a_1	a_2	...	a_n
b_1	x	x	x	x
b_2	x	x	x	x
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
b_m	x	x	x	x

b.

a_1	x
a_2	x
$\vdots$	$\vdots$
a_n	x

b_1	x
b_2	x
$\vdots$	$\vdots$
b_m	x

The single-agent case is shown in Table 1a. The action space is $A \times B$ with $a_i \in A$ and $b_j \in B$. With this space the learning agent has $n \cdot m$ values to store and to search through when selecting the action with the highest Q-value. In Table 1b, the action spaces of two agents are shown, each belonging to one actuator, A for the first agent and B for the second. The two action spaces together have $(m + n)$ values. This means that memory requirements and computation time during action selection are reduced by a factor of:

$$\frac{m \cdot n}{m + n} \tag{15}$$

If we set $m = n$, as we will do in the robotic systems in the next sections, the gain is a factor $\frac{1}{2}m$. When extending this approach to i actuators this gain increases to $\frac{1}{i}m^{i-1}$. Adding actuators to this MARL approach requires a linear increase in memory and computational time instead of an exponential increase. Therefore, Action Decomposition makes MARL more scalable in the number of actuators.

2.7 The state of the art

In Section 2.1 we explained that each Independent Learner has a non-Markovian problem to solve. This is because each IL is not aware of the actions of all other ILs. The system with all agents combined, the MMDP, does possess the Markov-property.

We propose to use these Independent Learners within one robot. With action decomposition (see Section 2.6) an MDP is split up into an MMDP, with one actuator per agent. Most recent literature that involves MARL is about multiple, often homogeneous robots having to work together to solve a problem. This leads to a system where it is hard to have full state information of all robots; a problem we do not have in single-robot systems. For example in [13], many cooperative multi-agent systems are described and good empirical results are achieved.

Theoretical literature on heterogeneous ILs mainly focuses on single state MMDPs with two or three actions available to each agent [4, 9, 14, 15]. The problems faced and sometimes solved in these papers may also (at least to some extent) occur in our proposed method. Because two or more agents have to work together in a stateless task, in literature most problems are described in game theory terminology. Because the learning problem for each agent is not stationary (see Section 2.1) each agent is faced with a moving target learning problem: the best policy depends on the other agents' policies. This problem is called the coordination problem [4]. In [4] the difficulties of ILs are investigated and it is shown that there are two basic problems. They are summarized below for the deterministic MMDP and the stochastic MMDP.

2.7.1 Deterministic MMDP

In the deterministic case, two situations give rise to problems. Since the cross terms in the joint action space are not stored, problems arise in situations like the 'penalty game' and the 'climbing game' [4].

Table 2. Reward function of a. Penalty Game b. Climbing game

a.

	a_1	a_2	a_3
b_1	10	0	k
b_2	0	0	0
b_3	k	0	10

b.

	a_1	a_2	a_3
b_1	10	-30	0
b_2	-30	7	6
b_3	0	0	5

In the 'penalty game', see Table 2a, k is a penalty. As can be seen two Nash-Equilibria [12] exist at (a_1, b_1) and at (a_3, b_3). A Nash-Equilibrium is an equilibrium in the joint action space A, such that each agent's individual action (in this case a_n and b_n) is a best response to the other's [12]. The reward received when joint action (a_1, b_1) is taken is the same as with (a_3, b_3). This means that if the learning episode converged, the chances of choosing action a_1 or a_3 are both 50%. The same applies for b_1 and b_3, giving rise to a 25% chance of choosing action (a_3, b_1) and a 25% chance of choosing (a_1, b_3), leading to a penalty of k. To make sure the actions that result in a penalty are not chosen, and both agents prefer to choose the same Nash-Equilibrium, some coordination might be needed.

In the climbing game (see Table 2b.) also two Nash-equilibria exist, an optimal one at (a_1, b_1) and a suboptimal one at (a_2, b_2). Because of the high penalties at (a_1, b_2) and (a_2, b_1) of -30 the expected reward for each agent if the other agent performs a random action is $\frac{10-30}{3}$ and negative. The same applies for (a_2, b_2), however that equilibrium can be reached starting from (a_3, b_3) and moving up to (a_3, b_2) and left to (a_2, b_2).

In [9] the IL is adjusted to cope with those problems. Instead of the action value function update rule in (4) an 'optimistic' assumption is made:

$$Q(s_t, a_t) \leftarrow \begin{cases} Q(s_t, a_t) + \alpha\delta, & \delta > 0 \\ Q(s_t, a_t), & \delta \leq 0 \end{cases} \tag{16}$$

This takes care of the problem in the climbing game. In [9], they keep track of the first best policy in each agent to make sure in the penalty game both agents choose the same (the first tried) optimum. With these two additions, convergence to the optimal solution is again guaranteed under the same conditions as with the single-agent MDP [9].

2.7.2 Stochastic MMDP; Lenient Learning

Within a stochastic MMDP, the fluctuations of the δ in a state-action pair due to the stochastic transitions cannot be distinguished from the fluctuations due to influence of the other agents' actions. This leads to an overestimation of the total expected reward when using the 'optimistic' assumption of (16) resulting in a loss of convergence.

In [14] Lenient Learning is proposed. This is a combination of the update rule in Equation (4) and (16). In the beginning of the learning trial the optimistic assumption is made to make sure optimal equilibria are found. After these are discovered, the lenience towards the other agents is tuned down returning the update to the original function (4). This transition is smoothly made with a Boltzmann probability function:

$$\begin{array}{c} x = \text{random}([0,1]), \\ Q(s_t, a_t) \leftarrow \begin{cases} Q(s_t, a_t) + \alpha\delta, & \delta > 0 \text{ or } x > \ell \\ Q(s_t, a_t), & \delta \leq 0 \text{ and } x \leq \ell \end{cases} \end{array} \tag{17}$$

with $x \in [0, 1]$ a random variable, and the state-action pair dependent lenience $\ell(s_t, a_t)$ defined as:

$$\begin{array}{c} \ell(s, a) = 1 - e^{-\kappa * \tau(s,a)}, \\ \tau(s, a) \leftarrow \beta\tau(s, a), \end{array} \tag{18}$$

where κ is the lenience parameter and $\tau(s, a)$ the lenience temperature of the state-action pair (s, a), that decreases with a discount factor $\beta \in [0, 1]$, each time the state-action pair is visited. We have extended Lenient Learning to a multi-state method with eligibility traces and applied it as a first test to the two-link manipulator (see Section 3).

2.8 Implications for our robots

Because convergence to the optimal solution is not guaranteed for the 'penalty game' and 'climbing game' (see Table 1), care needs to be taken when implementing ILs. In the test setups used in this paper, we expect that these two situations, however, will not occur very often in the joint action space.

With our policy (see Section 2.4) in combination with a random action-value function initialization, one of the two (or more) equally optimal Nash-equilibria of the penalty game will be chosen, at least at first. When one of the optimal equilibria is chosen and a reward was given for it, the action value function at that equilibrium is highest for all agents. This means unless both agents explore simultaneously and choose the other equilibrium, the action value function at the first equilibrium will be highest. Even when both agents explore to another optimal equilibrium, the action value function at the second equilibrium, at least the first few times the agents explore there, will be lower because of the learn rate, time and trace discounting factors that apply on the update (see Equation (7)). Even if, after many explorations to a second optimal equilibrium, action value functions at both equilibria are almost the same, they still will not be exactly the same because of the random initialization. Thus, unless the agents learn for a (almost) infinite time the two equilibria will not be valued the same.

If, in some states, a climbing-like game occurs, converging to the more robust but suboptimal equilibrium ((a_2, b_2) in Table 1b.) [15] instead of the the optimal one, does not necessarily effect performance much. In most cases the robot will reach its goal slightly slower than optimally. Because of the many (other) states that are visited before completion of the task this affect will be small.

To help solve the possible convergence problems discussed in this Section and in 2.7.2, Lenient Learning [14] is also implemented.

3 TWO-LINK MANIPULATOR

3.1 Model

The first test setup for our proposed method is a two-link manipulator [3], depicted in Figure 2. The system has two rigid links, which are connected by a motorized joint. One end of the system is attached to the world, also with a motorized joint. The system moves in the two dimensional horizontal plane without gravity according to the following fourth-order non-linear dynamics:

$$M(\alpha)\ddot{\alpha} + C(\alpha, \dot{\alpha})\dot{\alpha} = \tau \tag{19}$$

in which $\alpha = [\alpha_1, \alpha_2]$ and $\tau = [\tau_1, \tau_2]$. The mass matrix $M(\alpha)$ and the Coriolis and centrifugal forces matrix $C(\alpha, \dot{\alpha})$ have the following form:

$$M(\alpha) = \begin{bmatrix} P_1 + P_2 + 2P_3 \cos\alpha_2 & P_2 + P_3 \cos\alpha_2 \\ P_2 + P_3 \cos\alpha_2 & P_2 \end{bmatrix} \tag{20}$$

$$C(\alpha, \dot{\alpha}) = \begin{bmatrix} b_1 - P_3\dot{\alpha}_2 \sin\alpha_2 & -P_3(\dot{\alpha}_1 + \dot{\alpha}_2)\sin\alpha_2 \\ P_3\dot{\alpha}_1 \sin\alpha_2 & b_2 \end{bmatrix} \tag{21}$$

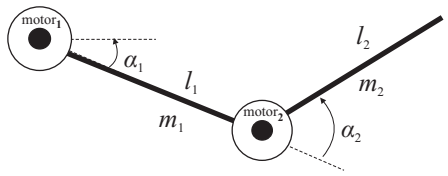


Figure 2. Two-link manipulator

Table 3. Physical parameters of the two-link manipulator

Symbols and values	Description
$l_1 = l_2 = 0.4$ m	link lengths
$m_1 = 1.25$ kg, $m_2 = 0.8$ kg	link masses
$I_1 = 0.066$ kgm^2, $I_2 = 0.043$ kgm^2	link inertias
$c_1 = c_2 = 0.2$ m	center of mass for the links
$b_1 = 0.08$ kg/s, $b_2 = 0.02$ kg/s	damping in the joints
$\tau_{1,max} = 1.5$ Nm, $\tau_{2,max} = 1.0$ Nm	maximum motor torques
$\dot{\alpha}_{1,max} = \dot{\alpha}_{2,max} = 2\pi$ rad/s	maximum angular velocities

Using the physical parameters from Table 3, we can calculate the parameters P_1, P_2 and P_3 in (20) and (21):

$$\begin{aligned} P_1 &= m_1 c_1^2 + m_2 l_1^2 + I_1 \\ P_2 &= m_2 c_2^2 + I_2 \\ P_3 &= m_2 l_1 c_2 \end{aligned} \tag{22}$$

Equation (19) is numerically integrated using the Runge-Kutta algorithm with time step $T_i = 0.01$s.

3.2 Learning

In our learning setup, we define two agents, one for each motor. Both agents get full state information, but no information about the other agent's action:

$$\begin{aligned} &\text{Agent 1: } S^1 = \alpha_1 \times \alpha_2 \times \dot{\alpha}_1 \times \dot{\alpha}_2, \quad A^1 = \tau_1 \\ &\text{Agent 2: } S^2 = \alpha_1 \times \alpha_2 \times \dot{\alpha}_1 \times \dot{\alpha}_2, \quad A^2 = \tau_2 \end{aligned} \tag{23}$$

The task of the system is to accomplish $\alpha_1 = \alpha_2 = \dot{\alpha}_1 = \dot{\alpha}_2 = 0$ as fast as possible. To this end, a reward is given when the angles and angular velocities are within a small region around 0: $|\alpha| < 0.17$ and $|\dot{\alpha}| < 0.2$. Both agents have to perform the right actions to trigger this reward and both agents will receive the same reward in that case. Furthermore, each time step a time penalty is given. The reward function r for performing action a_t at time t, equal for both agents, becomes:

$$r_t^1 = r_t^2 = \begin{cases} 100, & \text{if } |\alpha_t| < 0.17 \text{ and } |\dot{\alpha}_t| < 0.2 \\ -1, & \text{all other cases (time penalty)} \end{cases} \tag{24}$$

The agents perform a Q-learning update rule simultaneously at each time step according to (7). The time between each learning step $T_s = 0.05$s. The learn rate $\alpha = 0.4$, the exploration rate $\epsilon = 0.05$, the discount factor $\gamma = 0.98$ and the trace discount factor $\lambda = 0.92$. Both agents have their own tile coding function approximator with 16 tilings to approximate the Q-function. The action space is discrete; for every $Q^i(s, a_i)$ there is a separate function approximator so that there is no generalization between actions. The tile widths for the state space are 1/12 (rad) in the α_1 and α_2 dimensions and 1/6 (rad)s^{-1} in the $\dot{\alpha}_1$ and $\dot{\alpha}_2$ dimensions. The learning results will be compared with the single-agent case in which one agent has the full action space $A = \tau_1 \times \tau_2$.

3.3 Results

In order to test the learning performance of the two-link manipulator task, we defined a test set of 15 different initial conditions from which the manipulator has to complete its task. We regularly let the system perform the tasks from the test set and monitor the number of successfully completed tasks. During test runs, exploration is disabled. In the next sections, we test various learning algorithms and settings and compare their performance. The learning process is repeated for several different random initializations of the action-value function. In each result plot, the results are surrounded by 95% confidence interval graphs or error bars.

3.3.1 Single-agent vs. multi-agent Q(λ)-learning

In this test we compare the learning result of the two-link manipulator task in the single-agent case (SA) with the multi-agent case (MA), where all agents use Q(λ)-learning. Each actuator's action space is discretized into 7 steps. The result can be found in Figure 3. The learning curve of the SA case is slightly, but barely better than the MA case. This suggests learning behavior between a single-agent and our multi-agent algorithm is comparable.

A comparison of the difference in memory usage between the SA case and the MA case, for several sizes of the action space, can be found in Figure 4. Because we use an equal number of discrete actions m for both agents, according to (15) we can expect a memory reduction of $\frac{m}{2}$ in the multi-agent case compared to the single-agent case. As can be seen from the figure, the simulations follow the expected memory reduction factor. A comparison of the calculation

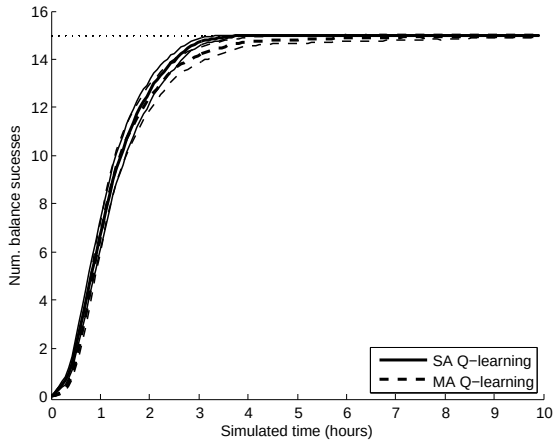


Figure 3. Single-agent (SA) Q(λ)-learning compared with multi-agent (MA) Q(λ)-learning for the two-link manipulator task (average over 400 independent runs).

time of an average learn step can be found in Table 4. It follows the same trend as for the memory reduction, with a bias for the overhead of other calculations than the best action search.

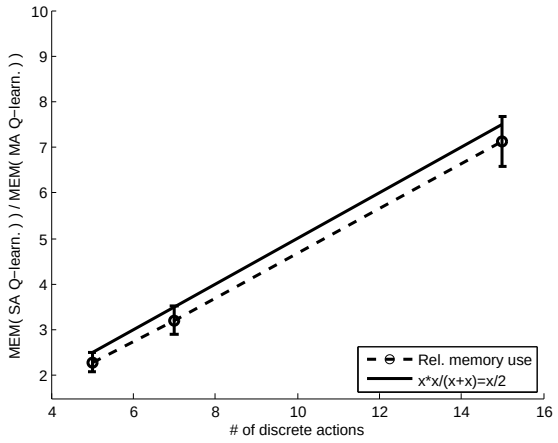


Figure 4. Memory usage comparison between the single-agent (SA) Q(λ)-learning setup and the multi-agent (MA) Q(λ)-learning setup for the two-link manipulator task, for several sizes of the action space (equal for both agents) (average over 25 independent runs).

Table 4. Single-agent (SA) vs. multi-agent (MA) Q(λ)-learning: relative calculation time of the learning step for the two-link modulator with m actions for each agent.

Two-link man.	$m = 5$	$m = 7$	$m = 15$
SA/MA	2.1	3.2	7.1

3.3.2 SARSA(λ)-learning

It could be that choosing SARSA for each agent makes the agent better able to deal with the changing policy of the other agent. A comparison between the single-agent and the multi-agent case, both using SARSA(λ)-learning, can be found in Figure 5. With SARSA the SA case is just like with Q(λ)-learning a little better than the MA case. Furthermore, the effect of using SARSA instead of Q-learning in the MA case can be seen in Figure 6. There is no significant difference between SARSA and Q-learning.

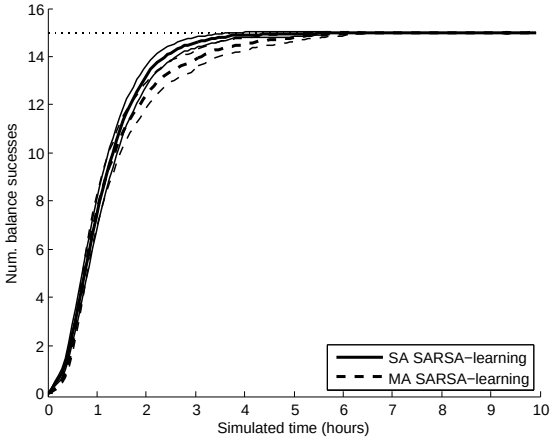


Figure 5. Single-agent (SA) SARSA(λ)-learning compared with multi-agent (MA) SARSA(λ)-learning for the two-link manipulator task (average over 400 independent runs).

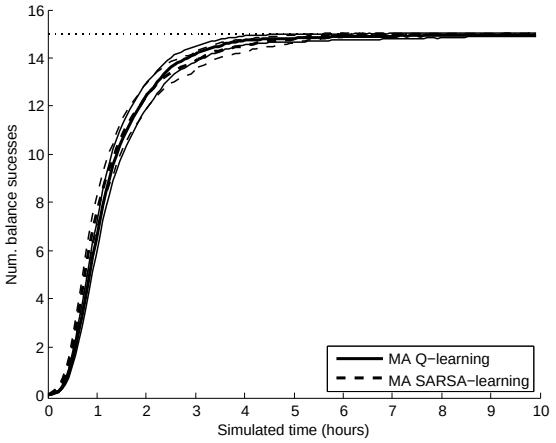


Figure 6. Multi-agent (MA) Q(λ)-learning compared with multi-agent (MA) SARSA(λ)-learning for the two-link manipulator task (average over 400 independent runs).

3.3.3 Synchronizing explorative actions

In the Independent Learner MA setup, all agents explore independently (13). However, it could be beneficial to synchronize exploration by letting both agents solely perform explorative actions simultaneously (14). Perhaps the chances of escaping from a local maximum (for instance in situation like the 'climbing game' in Table 1b.) increase when all agents deviate from their current policy simultaneously. We compared the independent exploration strategy with the synchronized exploration strategy. The result can be found

in Figure 7. It however, did not result in significantly better results. This could be because in our state-action value function there are not many non-optimal (Nash-)equilibria.

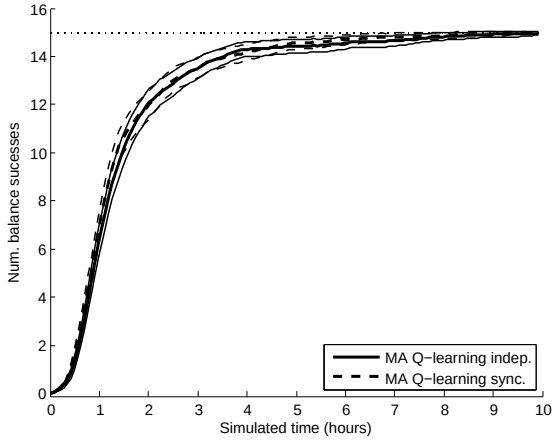


Figure 7. Multi-agent (MA) Q(λ)-learning with independent exploration vs synchronized exploration between the agents (average over 400 independent runs).

3.3.4 Lenient learning

As explained in Section 2.7, Lenient Learning might help the learning process by ignoring negative value updates in the beginning of learning. This would prevent one mistake of an agent to discourage the right policy of the other agent. A comparison between multi-agent Q(λ)-learning and multi-agent Lenient Q(λ)-Learning, with a lenience factor $\ell(s, a)$ for each state-action pair, with $\kappa = 2.0$ and a temperature discount factor $\beta = 0.95$, discounting at each visit, can be found in Figure 8. As can be seen a significant improvement is achieved. Tweaking of the lenience parameters could potentially increase this improvement even further. This subject deserves further attention in our future research.

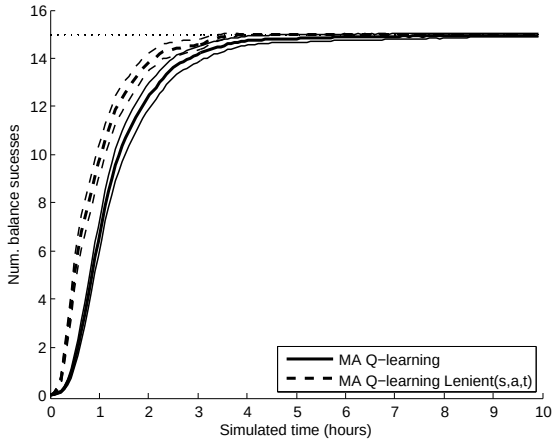


Figure 8. Multi-agent (MA) Q(λ)-learning compared with multi-agent (MA) Q(λ)-learning with Lenience for the two-link manipulator task (average over 400 independent runs).

4 META: A BIPEDAL WALKING ROBOT

4.1 Model

Our second test setup is the simulation of a bipedal walking robot, based on the prototype META [18], see Figure 9. META's construction is based on the concept of limit cycle walking [7]. With the concept of limit cycle walking, it is possible to construct a fully passive walking robot that can walk down a shallow slope (for energy input) without any actuation or control [5, 11], by carefully choosing the mass distributions and leg lengths. By adding actuation, the robustness increases and the energy input can come from an external source so that it can walk on flat terrain. Because META was designed according to the limit cycle walking concept, walking is a natural movement for the robot. META is effectively a 2D walking robot by using two pairs of parallel legs, which remove the sideways stability problem. The version of META that was modeled in [18] had one hip motor and a special mechanical construction that always kept the upper body upright, at an angle that bisects the angle between both upper legs. The prototype has recently been modified and now has two hip motors. One motor controls the joint between the upper body and the left upper leg, the other motor controls the joint between the upper body and the right upper leg. Each motor can apply a torque to its joint between -10Nm and $+10$Nm. The prototype was modeled

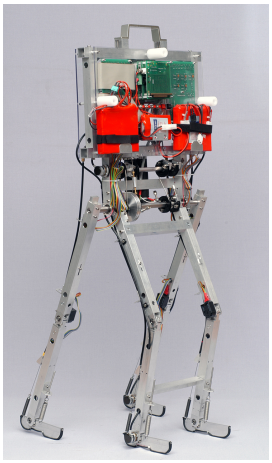


Figure 9. The bipedal walking robot META.

in the Open Dynamics Engine rigid body simulator [20] as a 7-link 2D model, see Figure 10. The joints are modeled by stiff spring-damper combinations. The knees are provided with a hyperextension stop and a locking mechanism which is released just after the start of the swing phase (i.e. right after making a footstep). Contact between the foot and the ground is also modeled by a tuned spring-damper combination which is active whenever part of the foot is below the ground. The model of the foot mainly consists of two cylinders at the back and the front of the foot. A set of physically realistic parameter values were derived from the prototype, see Table 5.

4.2 Learning

The full state space of the robot consists of the angle and angular velocity of all seven body parts, i.e. 14 state dimensions. Because the feet have relatively small masses and inertias, we assume that the

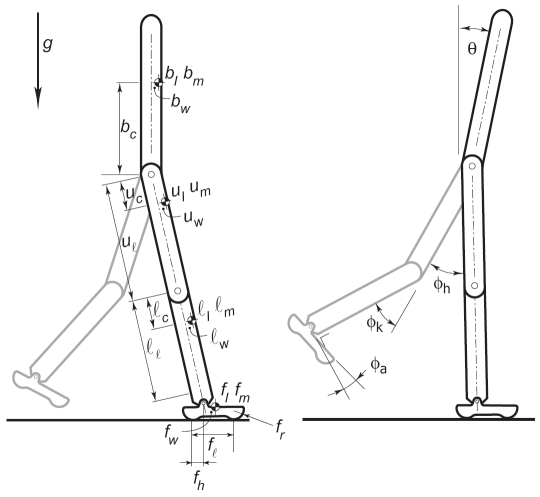


Figure 10. Two-dimensional 7-link model of META. Left the parameter definition, right the Degrees of Freedom (DoFs). Only the DoFs of the swing leg are given, which are identical to the DoFs of the other leg.

Table 5. Physical parameters of the model of META.

	Body(b)	Up.leg(u)	Lo.leg(ℓ)	Foot(f)
Mass m [kg]	8	0.7	0.7	0.1
Mom. of in. I [kgm^2]	0.11	0.005	0.005	0.0001
Length l [m]	0.45	0.3	0.3	0.06
Vert. offset CoM c [m]	0.2	0.15	0.15	0
Hor. offset CoM w [m]	0.02	0	0	0.015
Foot radius f_r [m]	-	-	-	0.02
Foot hor. offset f_h [m]	-	-	-	0.015

state transitions and the rewards of the system do not significantly depend on the angles and angular velocities of the feet. Therefore, we do not include them in the state space, which leaves us with 10 input dimensions. We assign a separate learning agent to each motor, so that we have a multi-agent system with two agents. Each agent has the same state space consisting of these 10 input dimensions and an action space consisting of one motor torque, discretized in 7 steps between -10Nm and +10Nm.

The task of META is to learn to walk with the highest possible forward velocity by actuating both hip motors. This requires the simultaneous coordination of the legs to make correct footsteps, as well as the coordination of the upper body; a task that is much more difficult than the learning task solved in [18]. The following rewards are used. Whenever the robot makes a footstep, a reward is given that is proportional to the length of the footstep. Together with a time discount factor γ close to 1, the robot will optimize for progressing as many meters in as little time as possible (however too large footsteps will lead to falling). A footstep is defined as the moment when the foot of the swing leg touches the ground while the hip angle is between 0.1 and 0.61 rad. These values are the minimum and maximum size of a step that allow walking in our model. Furthermore, a penalty is given when the robot falls. The rewards are based on the performance of the system as a whole, not on the behavior of a single-agent. To make a footstep, cooperation between both agents is required. Both agents have the same reward function r_t:

$$r_t^1 = r_t^2 = \begin{cases} 500/\text{m}, & \text{if footstep made, } 0.1 < |\varphi_{hip,t}| < 0.61 \\ -10, & \text{if the robot falls} \end{cases} \quad (25)$$

The agents perform a Q-learning update rule simultaneously at each time step according to (7). The time between each learning step $T_s = 0.018$s. The learn rate $\alpha = 0.5$, the explore rate $\epsilon = 0.05$, the time discount factor $\gamma = 0.995$ and the trace discount factor $\lambda = 0.92$. Both agents have their own tile coding function approximator with 16 tilings. Generalization is present between states as well as between actions. The tile widths are the same for all agents; for the upper leg angles: 1/6 rad, the upper leg angular velocities: 1/2 rad·s^{-1}, the lower leg angles: 1/2.1 rad, the lower leg angular velocities: 1/1.65 rad·s^{-1}, the body angle: 1/5.5 rad, the body angular velocity: 1/1.2 rad·s^{-1} and the output torque: 5 Nm.

The learning results will be compared with the single-agent case in which one agent has the full action space $A = \tau_1 \times \tau_2$.

4.3 Results

In order to test the learning performance of META's walking task, we regularly perform a walking run and monitor the number of footsteps the robot made. When 16 footsteps are made, we assume that walking has succeeded and we end the trial. Therefore, the maximum performance of a test run is 16 footsteps. During test runs, exploration is disabled. In the next sections, we test various learning algorithms and settings and compare their performance. The learning process is repeated for several different random initializations of the action-value space. In each result plot, the results are surrounded by 95% confidence interval graphs. In contradiction to the two-link manipulator of Section 3, for Meta, Lenient Learning has not yet been successfully applied. This will be done in future research.

4.3.1 Single-agent vs. multi-agent Q(λ)-learning

In this test we compare the learning result of META's walking task in the single-agent case (SA) with the multi-agent case (MA), where all agents use Q(λ)-learning. The result can be found in Figure 11. First of all, we can conclude that the walking task is successfully learned in both the single-agent Q(λ)-learning and multi-agent Q(λ)-learning case. The robot is able to walk after about 15 hours. Second, we can conclude that the difference in learning speed between the SA case and the MA case is negligible. However, in the end, the performance in the multi-agent case is slightly less than in the single-agent case.

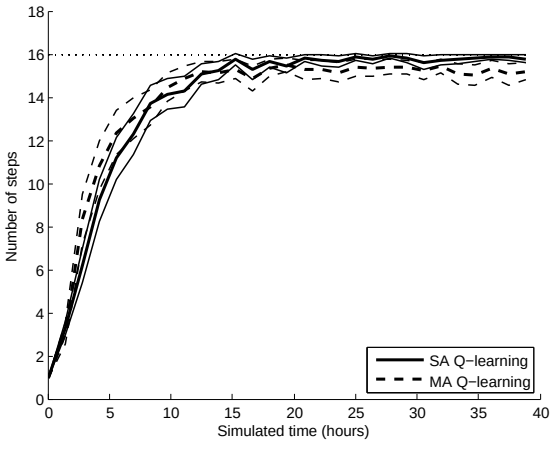


Figure 11. Single-agent (SA) Q(λ)-learning compared with multi-agent (MA) Q(λ)-learning for META (average over 96 independent runs).

4.3.2 Multi-agent SARSA(λ)-learning vs Q(λ)-learning

Because SARSA is an on-policy learning algorithm, it could be the case that when the agents learn with SARSA, each agent is better able to deal with the changing policy of the other agent. A comparison between the MA case using SARSA(λ)-learning and the MA case using Q(λ)-learning can be found in Figure 12. The learning curves do not differ significantly and the SARSA algorithm has no added value in this case.

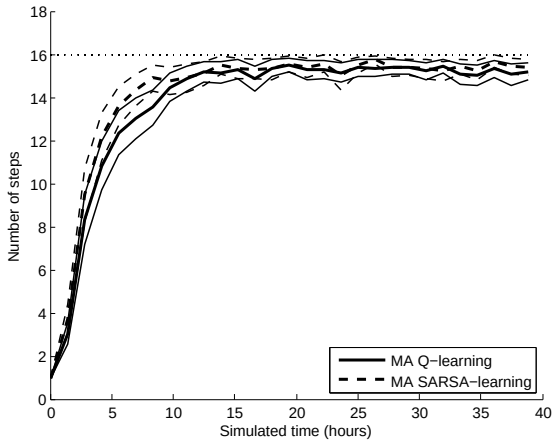


Figure 12. Multi-agent (MA) Q(λ)-learning compared with multi-agent (MA) SARSA(λ)-learning for META (average over 96 independent runs).

4.3.3 Synchronizing explorative actions

In the Independent Learner MA setup, all agents explore independently (13). However, it could be beneficial to synchronize exploration by letting both agents solely perform explorative actions simultaneously (14). Perhaps the chances of escaping from a local maximum as in the 'climbing game' of Section 3 increase, when all agents deviate from their current policy simultaneously. We compared the independent exploration strategy with the synchronized exploration strategy. The result can be found in Figure 13. The learning graphs do not differ significantly. In this case, synchronization of explorative actions does not increase learning speed or performance.

4.3.4 Reduced state spaces

Now that we have created a multi-agent setting in which each agent controls one actuator, it might be the case that each agent does not need to know the full state of the robot in order to control its own actuator. Note that we then violate the Markov property not only because all agent specific transition functions are time-dependent (see Equation (3)), but also because we have hidden state variables. In this test, the agent that controls the joint between the upper body and the stance leg does not include the angular velocity of the lower swing leg in its state space. The agent that controls the joint between the upper body and the upper swing leg still uses the full state space of 10 dimensions. The result can be found in Figure 14. From this graph, it is clear that the performance of the reduced state space case is far less than the original MA case, but not failing completely. It is clear

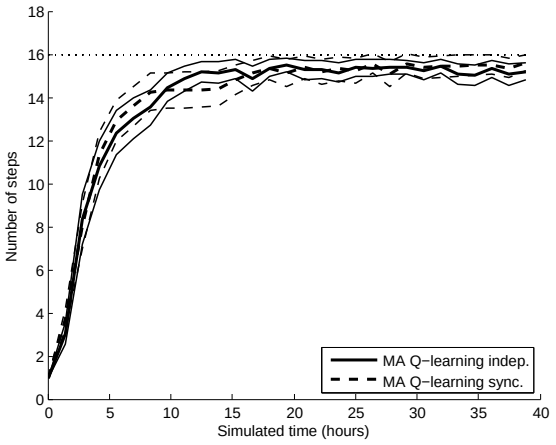


Figure 13. Multi-agent (MA) Q(λ)-learning with independent exploration vs synchronized exploration between the agents (average over 96 independent runs).

that divergence issues occur later on in the graph. Apparently, the incomplete state space is violating the Markov property too much.

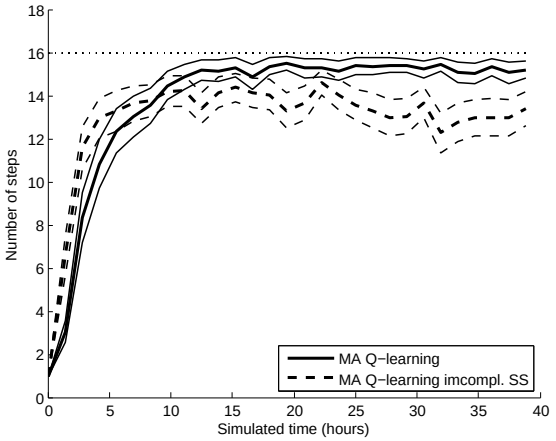


Figure 14. Multi-agent (MA) Q(λ)-learning. The original MA setting is compared with a MA setting in which one agent has an incomplete state space (average over 96 independent runs).

5 DISCUSSION

In this paper, we used a multi-agent approach to make reinforcement learning more scalable for learning complicated robotic tasks. When adding actuators to a learning agent, its action space increases exponentially, requiring a longer learning time, more storage space and more computational search time during action selection. Decomposing the combined action space of these actuators by splitting it up over different agents solves the last two problems: instead of exponentially increasing in size of the action space, it now becomes linear in the number of actuators. Unfortunately our approach is not proven to converge to the optimal solution. Current theory on convergence, focusing on single state problems, shows two obstacles for convergence: two typical situations called the penalty game and the climb-

ing game. We showed however that in our simulations these problems didn't result in bad performance.

We implemented the Independent Learner algorithms proposed for two simulated robots: a two-link manipulator and Meta, a bipedal walking robot. The two-link manipulator had to learn to get its two links in a stable position with angles of both links at 0. Meta had to learn to walk and to stabilize its upper body. Both systems have two actuators and thus two agents in the multi-agent approach. In a direct comparison between single-agent and multi-agent Q-learning, the performance of cooperative, heterogeneous independently learning agents was not very different from the single-agent case, while memory requirements and action selection computation time decreased with a factor $\frac{m \cdot n}{m+n}$, in which m and n are the number of discrete actions per actuator.

With single-agent learning, SARSA and Q-learning performed equally well in both the two-link manipulator and Meta. Because SARSA has an on-policy update rule, it might perform better than Q-learning with multi-agent learning. SARSA learning was implemented in both robots and resulted, however, in similar learning performance to Q-learning.

We proposed an idea to quickly get out of local, suboptimal (Nash)equilibria; to make sure both agents in each test setup do an ϵ-greedy exploration simultaneously. However, this did not result in significantly different results. This could be because in our state-action value function there are not many non-optimal (Nash)equilibria.

In recent literature the Lenient Learning algorithm was proposed as a method to quickly overcome the problems with unpredictable and suboptimal other players. IL was implemented for the two-link manipulator and a significant improvement was found in learning speed and performance. Tweaking of the lenience parameters could potentially further increase this improvement. For Meta, good results with Lenient Learning have not yet been achieved. This subject deserves further attention in our research.

6 CONCLUSIONS

In this paper we showed that using multiple cooperative, heterogeneous independently learning agents, each controlling one actuator of a robot, is a promising method of making reinforcement learning more scalable in the number of outputs and a better candidate for learning in complex robots. The learning performance is similar to the single-agent case, however, the computational time needed to complete learning and the amount of memory needed to store the state-action space are significantly decreased; from an exponential problem in the number of actuators, it became a linear problem. In addition, we showed that Lenient Learning significantly increased learning speed in one of our test setups. Overall, this makes our proposed method better than single-agent learning, at least for the two double-actuator robotic systems that were tested. The method is also very suitable for implementation on a multi-processor or multi-computer system. Future work will concentrate on testing the method on robotic systems with more actuators. Also, in the near future we will test this approach on a real robot with multiple actuators.

REFERENCES

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

[2] C. Boutilier, 'Planning, learning and coordination in multiagent decision processes', *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, 195–210, (1996).

[3] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska, 'Fuzzy Approximation for Convergent Model-Based Reinforcement Learning', *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, 1–6, (2007).

[4] C. Claus and C. Boutilier, 'The dynamics of reinforcement learning in cooperative multiagent systems', *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, **746**, 752, (1998).

[5] SH Collins, M. Wisse, and A. Ruina, 'A two legged kneed passive dynamic walking robot', *Int. J. of Robotics Research*, **20**(7), 607–615, (2001).

[6] G.J. Gordon, 'Reinforcement learning with function approximation converges to a region', *Advances in Neural Information Processing Systems*, **13**, 1040–1046, (2001).

[7] D.G.E. Hobbelen, *Limit cycle walking*, Delft University of Technology, 2008.

[8] N. Kohl and P. Stone, 'Policy gradient reinforcement learning for fast quadrupedal locomotion', *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, **3**.

[9] M. Lauer and M.A. Riedmiller, 'An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems', *Proceedings of the Seventeenth International Conference on Machine Learning table of contents*, 535–542, (2000).

[10] M.L. Littman, 'Markov games as a framework for multi-agent reinforcement learning', *Proceedings of the Eleventh International Conference on Machine Learning*, **157163**, (1994).

[11] T. McGeer, 'Passive Dynamic Walking', *The International Journal of Robotics Research*, **9**(2), 62, (1990).

[12] J.F. Nash Jr, 'Non-Cooperative Games', *Annals of Mathematics*, **54**(2), (1951).

[13] L. Panait and S. Luke, 'Cooperative Multi-Agent Learning: The State of the Art', *Autonomous Agents and Multi-Agent Systems*, **11**(3), 387–434, (2005).

[14] L. Panait, K. Sullivan, and S. Luke, 'Lenience towards Teammates Helps in Cooperative Multiagent Learning', *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems–AAMAS-2006, ACM Press, New York*, (2006).

[15] Liviu Panait, Karl Tuyls, and Sean Luke, 'Theoretical advantages of lenient learners: An evolutionary game theoretic perspective', *Journal of Machine Learning Research*, **9**(Mar), 423–457, (2008).

[16] J. Peters, S. Vijayakumar, and S. Schaal, 'Reinforcement learning for humanoid robotics', *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots*, (2003).

[17] D. Precup, R.S. Sutton, and S. Dasgupta, 'Off-policy temporal-difference learning with function approximation', *Proceedings of the Eighteenth International Conference on Machine Learning*, 417–424, (2001).

[18] E. Schuitema, DGE Hobbelen, PP Jonker, M. Wisse, and JGD Karssen, 'Using a controller based on reinforcement learning for a passive dynamic walking robot', *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, 232–237, (2005).

[19] S.P. Singh and R.S. Sutton, 'Reinforcement learning with replacing eligibility traces', *Machine Learning*, **22**(1), 123–158, (1996).

[20] R. Smith et al., 'Open Dynamics Engine', *Computer Software. From http://www.ode.org*, (2006).

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

[22] R. Tedrake, TW Zhang, and HS Seung, 'Stochastic policy gradient reinforcement learning on a simple 3D biped', *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, **3**.

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

[24] C.J.C.H. Watkins, *Learning from Delayed Rewards*, Cambridge University, 1989.

A Reinforcement Learning Approach for Production Control in Manufacturing Systems

A.S. Xanthopoulos, D.E Koulouriotis[1] and A. Gasteratos

Democritus University of Thrace, Hellas, email[1]: jimk@pme.duth.gr

Abstract. The problem of production control in serial manufacturing lines that consist of a number of unreliable machines linked with intermediate buffers is addressed. We make use of Reinforcement Learning methodologies in order to derive efficient control policies. Our aim is to derive control policies that are more state-dependent and therefore more efficient than well-known pull type control policies such as Kanban. Manufacturing systems of this type are studied under average measures such as average WorkInProcess inventories etc. and thus, a learning algorithm from the currently developing field of Average Reward Reinforcement Learning was applied. The Reinforcement Learning control policy was compared to three existing efficient pull type control policies, namely Kanban, Base Stock and CONWIP on the basis of simulated data and found to outperform them. The simulation experiments involved a single-product system with two machines that allows backordering. Numerical results are presented along with a qualitative interpretation of our findings. The paper concludes with directions for future research.

1. INTRODUCTION

The manufacturing process of a plethora of industrial end products is structured as a series of operations such as machining, forming etc. that sequentially take place on raw parts and sub-assemblies in order to form the finished product. The various manufacturing stations of the system that can be single machines, manufacturing cells etc. have non-deterministic production times and are susceptible to failure. The unreliability of the manufacturing operations along with the stochastic demand for final products dictates the use of safety buffers of intermediate and finished parts in order to attain the target service rate. However the use of safety stocks incurs significant holding costs that could bring the manufacturer to a position of competitive disadvantage and this is why a lot of effort has been put on the implementation of efficient methods for coordinating the manufacturing process.

A time-honored approach to modeling serial manufacturing lines is to treat them as Markov Processes (Gershwin, 1994, Veatch and Wein, 1992). The sequential decision-making task of coordinating production control can then be formulated as a Markov Decision Problem (see Puterman, 1994 and Littman et al, 1995). The solution to a Markov Decision Problem, (MDP), is a mapping from states to actions which is called optimal policy, that determines state transitions to maximize/minimize a properly defined performance criterion. There are well-known iterative algorithms for solving MDPs such as policy iteration (Howard, 1960), value iteration (Bellman, 1957) and variations of the two above mentioned techniques (asynchronous dynamic programming algorithms in Bertsekas, 1987). However the classic dynamic programming approach entails two major drawbacks: Bellman's famous curse of dimensionality and the need for a complete mathematical model of the underlying problem. The term dimensionality curse refers to the exponential computational explosion that takes place with the increase of the system state space rendering a wide range of realistic problems intractable by dynamic programming, (DP), algorithms. Another serious limitation posed by DP techniques is that they require transition probability and reward information in order to be employed in a certain problem. The later severely complicates the task of system modeling while in the same time rules out real-world problems where a complete model is simply not available.

Due to the above mentioned limitations, researchers have focused their efforts at developing sub-optimal yet efficient production control policies. A class of well-studied control mechanisms known as pull type control policies/mechanisms has come to be widely recognized as capable of achieving quite satisfactory results in serial manufacturing line management. These policies are the result of the emergence of the JustInTime/lean manufacturing philosophy and their main representative is the Kanban control policy that was originally developed by the Toyota Motor industry and became the topic of considerable research thereafter, (Sugimori et al, 1977, Buzacott and Shanthikumar, 1993, Berkley, 1992, Karaesmen and Dallery, 2000). Two other important and also well-studied pull control policies are the Base Stock (Buzacott and Shanthikumar, 1993) and the CONWIP (Spearman et al, 1990) control policies.

Pull type manufacturing control mechanisms as efficient as they can be, still remain sub-optimal heuristics with a number of serious inherent weaknesses, (e.g. the Kanban policy has limited flexibility when it comes to rapid reaction to incoming demands, the WorkInProcess inventory is unbounded in a Base Stock system), and require a lot of fine-tuning in order to perform well. Another important shortcoming is that pull control mechanisms do not take into consideration the operational status of the system's machines, that is whether a machine is in working condition or has failed and is currently under repair.

The writers' motivation was to utilize artificial intelligence methodologies in order to derive automatically dynamic, more state-dependent, near-optimal policies that would outperform the existing pull type policies. In our attempt to do so, we use a method from the field of Reinforcement Learning (Sutton and Barto, 1998, Kaelbling et al 1996). Related work can be found in Paternina-Arboleda and Das (2001) and Mahadevan and Theocharous (1998) but their learning algorithm is different than the one used here. Reinforcement Learning, (RL), can be viewed as a collection of techniques and learning algorithms for teaching agents optimal policies through interaction with their environment which is usually simulated by a computer program. Reinforcement Learning has received a lot of attention over the last few years as it provides the

means for solving many challenging problems, previously considered to be intractable. In this paper we applied a variation of R-learning (Schwartz, 1993), presented in Singh (1994) which uses average reward as its performance criterion, a choice that was dictated by the fact that we are interested in the average performance measures of the manufacturing system such as average WorkInProcess inventories, (WIP), etc. Average Reward Reinforcement Learning is still considered to be at an early stage with many important issues remaining open to discussion whereas RL algorithms that maximize average reward currently existing in the open literature are scarce. The first Average Reward RL algorithm was R-learning (Schwartz, 1993) followed by some variations of the original R-learning algorithm due to Singh (1994). Model-based average reward RL algorithms were developed by Mahadevan (1996) and Tadepalli and Ok (1998), Das et al. (1999) introduced SMART, an average reward algorithm for semi-Markov systems while Gosavi (2004) presented an RL algorithm for maximizing average reward based on policy iteration.

The RL control policy that we obtained by applying Singh's algorithm was compared to three existing efficient pull type control policies, Kanban, Base Stock and CONWIP on the basis of simulated data and found to outperform them in the task of minimizing WIP subject to a Service Level, (*SL*), constraint. The simulation experiments involved a single-product system with two machines that allows backordering. Analytical numerical results are presented along with a qualitative interpretation of our findings.

The remaining material of this paper is structured as follows. In Section 2 we offer the system description of a serial manufacturing line. Sections 3-3.3 are devoted to the presentation of the Kanban, Base Stock and CONWIP control policies. In Sections 4-4.2 we discuss the main aspects of the Reinforcement Learning framework and the algorithm that we applied in the underlying problem. We report our findings from the simulation experiments that we conducted for a two-station serial line in Sections 5-5.3. Finally, in section 6 we state our concluding remarks and point to possible directions for future research.

2. SYSTEM DESCRIPTION

In this paper we examine manufacturing systems that produce end products of a single type and consist of several manufacturing stations in series. A manufacturing station is a manufacturing/inventory module that incorporates one or more production operations (e.g. a single machine, a flexible manufacturing cell etc) grouped together and labeled as the manufacturing facility and a physical area of storage, the output buffer of that station. Production in batches is not allowed in this system as well as the reworking of parts in the same manufacturing station. The case of defective station *i* parts is also ruled out. Raw parts undergo production operations in the various manufacturing stations sequentially and are gradually converted into final products as they follow their downstream trajectory. By assumption, the system has an infinite supply of raw materials, so the first manufacturing facility is never starved. Demands arrive to the system at random time intervals and request the release of one part from the finished goods buffer. Provided that there is at least one finished part in the last buffer, the customer demand is satisfied instantaneously. If there are no final parts available, the demand is backordered. In the case where the system operates under a complete backordering policy and since customer impatience is considered to be absent in our model, no customer demand is lost to the system. Another option is the partial backordering policy where backorders are not allowed to exceed a predefined level. Manufacturing facilities are capable of processing only one part at a time. As soon as a station *i* part is produced, it is placed in the *i*th output buffer of the serial line where it waits for two conditions to be satisfied in order to be released to the next manufacturing facility: availability of the station *i* manufacturing station and authorization by the control policy. There is no delay in material handling between stations. Production control policies provide alternative ways of coordinating the release of parts from one station to another. Note that the points where production control is exerted by the controller are not fixed but depend on the control policy under which the system operates, e.g. one extreme condition is when production control is applied only to the first station as we will see in a subsequent section. All manufacturing facilities have random production time, time between failures and repair time. Time intervals between demand arrivals are also stochastic.

3. PULL PRODUCTION CONTROL POLICIES

Production control schemes that coordinate the production activities in a serial line based only on actual occurrences of demand are classified as pull type production control policies/mechanisms. Pull type control policies implement the JustInTime, (JIT), manufacturing philosophy and have attracted considerable attention over the past years as they are widely considered to outperform MRP-based production control systems. According to the JIT manufacturing philosophy a manufacturing system should maintain the minimum levels of safety stocks that are required in order to meet the target service level while in the same time has the ability to react rapidly to incoming orders. Pull production control policies are efficient heuristics with the major advantage of implementation simplicity that characterizes the host of them. The most important issue when applying pull type control of a certain type in a production system is to determine the best policy within this class of policies by appropriate control parameter selection. In sections 2.2 through 2.4 we present three fundamental pull control systems and namely, the Kanban (Sugimori et al, 1977), the Base Stock (Buzacott and Shanthikumar, 1993) and finally the CONWIP (Spearman et al, 1990) control mechanisms.

3.1 Kanban Control Policy

The Kanban production control policy was originally developed by the Toyota Motor Company in the mid-seventies and turned out to become almost a synonym for JIT manufacturing in the years to follow as it provided a conceptually clear, easy to implement and efficient way to coordinate the production process in a serial line. The philosophy of the Kanban control mechanism is quite simple. Each manufacturing station in a Kanban system has a fixed number of station *i* production authorizations; the system's control parameters which are set at design time. The K_i production authorizations (or kanbans) equal the maximum number of parts that are allowed in station *i*. Whenever a station *i* part exits its corresponding output buffer the controller authorizes the release of a station *i*-1 part into the next manufacturing facility. Thanks to this simple policy Kanban control achieves very tight coordination of the several production stations appearing in a system. However this happens in the expense of the manufacturing system's ability to respond swiftly to customer demands as the information of an arrival is transmitted across the system station by station through kanban authorizations. If there is a point where a part is not

available in order to be released to the next station, then no Kanban card is sent to the upstream production facility and the transmission of the information of the customer order is interrupted.

3.2 Base Stock Control Policy

The term base stock, also written as installation stock, is borrowed from inventory systems theory (Axsäter and Rosling, 1993), and refers to the initial levels of the systems intermediate and final product buffers. The initial base stock levels B_i are the system's only control parameters as the WIP is not constrained as in the Kanban control mechanism by a finite number of local production authorizations. The term WIP refers to the total number of parts that exist in the buffers of the manufacturing system. "Station *i* WIP" refers to the number of parts that are in the output buffer of this station. On the contrary, the WIP in a Base Stock system is not bounded and that is why this control policy can never be optimal as it was proven in Veatch and Wein (1994). The arrival of a demand is immediately transmitted to every manufacturing station in the system authorizing it to produce a new part. The theoretically strong point of this control mechanism is the high degree of responsiveness to the demand's fluctuations with the disadvantage of insufficient inventory control.

3.3 CONWIP Control Policy

CONWIP control was introduced in 1990 by Spearman et al. as a novel control mechanism for lean manufacturing, however as argued by Liberopoulos and Dallery (2000), it can be easily classified as a special case of the Kanban control system. In a CONWIP system production control is applied only to the first manufacturing station, whereas all the remaining stations have the perpetual authorization to produce provided that this is feasible. The CONWIP controller authorizes the first station to fabricate a new station 1 part as soon as a part exits the finished goods buffer in an effort to maintain the total WIP constant (CONstantWIP). An inherent characteristic of this control mechanism is that the WIP tends to accumulate in the last buffer. The system's sole control parameter is the maximum number of parts allowed in the last buffer or equivalently the number of CONWIP type production authorizations. A CONWIP manufacturing system can be seen as a one-station Kanban system where all of the serial line's production activities have been functionally aggregated in a single manufacturing facility.

4. FUNDAMENTALS OF REINFORCEMENT LEARNING

The majority of the published research in Reinforcement Learning is devoted to the following two models of optimal agent behavior; the maximization of the cumulative sum of rewards and the maximization of the discounted sum of rewards. The first case is appropriate for tasks that are naturally sub-divided in separate episodes while the second measure is used as a means to keep the infinite sum of rewards bounded in continual tasks where the decision-making process is repeated forever. Discounting future rewards makes perfect sense when it comes to problems encountered in e.g., economy, however this optimality criterion is not well-tailored for numerous Reinforcement Learning tasks. Many problems from queuing theory, manufacturing and other cyclical tasks are studied under average measures and therefore it is more logical to want to maximize the average reward per time step collected by the agent:

$$\lim_{n \to \infty} E\left(\frac{1}{n}\sum_{t=1}^{n} r_t\right) \tag{1}$$

A policy π^* that maximizes average payoff per time step is called gain-optimal. There are known issues concerning this optimality framework that can be alleviated by adopting a more generalized optimality criterion known as bias-optimality, (Kaelbling et al., 1996, Mahadevan, 1996) that also takes into consideration the reward gained in the initial phase of the agent's life. However for most practical purposes it is adequate to use the gain-optimal model in addition to the fact that bias-optimal algorithms are still considered to be experimental and significantly less well-understood.

A stationary policy π, is a mapping from states, $s \in S$, and actions $a \in A(s)$, to the probability $prob(s,a)$ of choosing action a when in state *s*, where *S* is the finite set of states in the task and *A(s)* the permissible actions in state *s*. A policy in general may be non-deterministic. Model-free Average Reward Reinforcement Learning algorithms are iterative stochastic approximation algorithms that use sample state transitions and sample rewards generated by simulation models in order to estimate relative value functions and average payoff. Informally, relative value functions are functions of states (or state action pairs) to real values that quantify the ''usefulness'' of being in a given state (or taking a certain action when being in a given state) when following a particular policy. Here, the term usefulness refers to the expected future rewards of the agent. Let ρ^π denote the average expected reward per time step under policy π:

$$\rho^\pi = \lim_{n \to \infty} E\left(\frac{1}{n}\sum_{t=1}^{n} r_t\right) \tag{2}$$

We define the relative value of state *s* under policy π as:

$$V^\pi(s) = E_\pi\left(\sum_{n=1}^{\infty} r_{t+n} - \rho^\pi \mid s_t = s\right) \tag{3}$$

Similarly, the relative value of taking action a in state *s*, or state-action value for short, is written as:

$$Q^\pi(s,a) = E_\pi\left(\sum_{n=1}^{\infty} r_{t+n} - \rho^\pi \mid s_t = s, a_t = a\right) \tag{4}$$

The solutions to the Bellman equation, (Equation 5), for average reward MDPs are the average payoff and the relative state-action values for the optimal policy π^*, denoted by ρ^* and Q^* respectively:

$$Q^*(s,a) = r(s,a) - \rho^* + \sum_{s' \in S} prob_{ss'}(a)\left[\max_{a' \in A(s')} Q^*(s',a')\right], \quad \forall s \in S, a \in A \tag{6}$$

where $prob_{ss'}(a)$ is the transition probability from state *s* to state *s'* when choosing action *a*, and $r(s,a)$ the reward on executing action *a* in state *s*.

4.2 Variation of R-Learning

The first average reward RL algorithm to appear in the literature was Schwartz's R-learning (1993), and since then only a few more algorithms were added to this category, (see Introduction for a listing of relevant algorithms). In this paper we used a variant of R-learning developed by Singh (1994) in order to derive efficient control policies for serial manufacturing lines. This improved version of R-learning is presented in the following Listing 1.

Listing 1. Variant of R-learning (Algorithm 3)

t=0, initialize ρ_t and $Q_t(s,a)$ for all *s, a*.

- Let *s* be the current state. Select $a \in A(s)$ according to action selection strategy, (e.g. e-greedy)
- Observe immediate reward r_{t+1} and next state s'
- $Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t[r_{t+1} - \rho_t + \max_{a'} Q_t(s',a')]$, $a' \in A(s')$
- $\rho_{t+1} = (1-\beta_t)\rho_t + \beta_t[r_{t+1} + \max_{a'} Q_t(s',a') - Q_t(s,a)]$, $a' \in A(s')$
- Decrease parameters α and β, and the exploration parameter *e*
- Loop

ρ_t and $Q_t(s,a)$ are approximations of the average reward and the state-action values, respectively, in epoch *t*. $A(s)$ is the set of admissible actions in state *s*. α_t is the learning rate for the Q-values and β_t is the learning rate for the average reward estimate, in decision making epoch *t*. *e* is the exploration parameter.

4.3 Methodology and Notation

In this section we discuss the way we fit the problem of coordinating production in a serial line into the Reinforcement Learning framework. First of all we must point out that there is a subtle distinction that needs to be drawn between the environment's relative position to the agent in the RL formulation and in the real world. In an actual manufacturing line the controller is embedded to the system whereas in the RL model the decision-making agent and the controlled system are two distinct entities interfaced together. The discrete set of states that the agent can find itself in is denoted by S and is given by the following expression:

$$S = \{(M_i, O_i, Bck) : i = 1,2,....,n\} \tag{7}$$

where *n* is the number of manufacturing stations and $M_i \in \{1,0,2\}$ are the possible conditions of the station *i* manufacturing facility. We denote failure in the manufacturing facility by 0, working condition by 1 and the state of being idle by 2. $O_i \in \{0,1,2,....\}$ is the station *i* buffer's state and finally, $Bck \in \{0,1,2,....\}$ is the number of backordered demands. The available actions to the controller are *authorize production* and *do not authorize production* of a new station *i* part for each of the *n* manufacturing stations, hence the number of admissible actions is 2^n. Let *A* denote the action space; as a result the complete state-action pair space is written as $S \times A$. The decision-maker interacts with the controlled system at discrete time steps $t = 1,2,3,...$ where the agent receives a representation of the environmental state and based on that information selects an action.
Each decision made by the agent has an associated cost which is given by Equation 7:

$$r_{t'} = -\sum_{i=1}^{n} q_i \overline{O_i} - c\overline{Bck} \tag{8}$$

where $\overline{O_i}$ and $\overline{Bck}$ are the time-averaged WIP in buffer *i* and level of backordered demands respectively until the next decision making epoch while q_i and c are positive constants. An action that yields relatively large amount of reward (or to be more precise low cost in our model) causes an improvement on the corresponding relative action value. In the algorithm that we implemented for the purposes of this paper state-action pair values are stored in a look-up table but in order to cope with large scale problems the use of some kind of a function approximation would be imperative. At each discrete time step *t* the agent implements a policy π_t on the basis of the relative Q-values of the state-action pairs. Most of the time the controller acts *greedily*, that is, it selects the action with the higher Q-value but in a fraction *e* of time it deviates from this behavior by selecting actions randomly in order to explore actively the state-action space. The agent explores intensively in the initial phase of the control task and then the amount of exploration slowly decays with time in order to allow convergence to the derived as optimal policy. The agent's goal which is conveyed to it through the cost assignment scheme as we will elaborate in the section to follow is to minimize WIP inventories subject to a service level constrain:

$$\min \sum_{i=1}^{n} \overline{WIP_i} \tag{9}$$

under the constraint:

Service Level>*t%*

where *t* is the service level target also known as the *fill rate* and specifies the proportion of customer orders which are satisfied instantaneously from the existing finished products stock.

5. SIMULATION

5.1 Simulation Case

The simulation experiments involved a two – station manufacturing line with equal operation times that operates under a policy that allows backorders. Machines operate with service rates which are normally distributed random variables with mean 1.0 parts/time unit and s.d. 0.01 ($R_p \sim N(1.0,0.1)$). Repair to failure times are exponentially distributed with mean 1000 time units. Failures are operation dependent. Repair times are also assumed exponential with a MTTR of 10 time units. Times between two successive customer arrivals are exponential random variables with mean 1.66 time units. Simulation time is set to 30000.0 time units. The objective is to minimize WIP while maintaining a 90% service level.

5.2 Experimental Setup

Prior to comparing the RL derived policy to the pull type heuristics we need to determine the best control parameters for each one of the Kanban, Base Stock and CONWIP policies. The parameters of the Kanban policy are the number of local production authorizations denoted by K_i while the Base Stock policy is characterized by the initial base stocks denoted by B_i. The parameters of a CONWIP system are the *C* global, (CONWIP type), production authorizations and the initial stock of intermediate and final parts. Note that by increasing the number of kanbans, base stocks or CONWIP type authorizations, the average throughput of the serial line increases and hence, the service level, but so does the WIP along with the related costs. In order to find the best parameters for each pull control policy we perform an incremental search over the space of feasible parameter sets. The results are displayed in Table 1.

Table 1. Pull production control policies parameter sets

	K_1 / B_1	K_2 / B_2	C
Kanban	1	4	-
Base Stock	0	5	-
CONWIP	2	3	5

The actions available to the agent are to authorize or not production in each one of the system's stations, therefore: *A*={[1 1],[1 0],[0 1],[0 0]}. In order to constrain the number of the system states we define the permissible buffer conditions to be $O_i \in \{0,1,2,3,4\}$ and we allow only partial backordering: $Bck \in \{0,1,2,3,4\}$. The default course of action when the inventory in a buffer rises beyond the allowable limit is to stop production in that certain station. The opposite happens when the number of backordered demands exceeds the predefined levels given that it does not violate the previous rule. The upper limit for the buffer levels is analogous to the control parameters of the pull type policies and the maximum number of unsatisfied demands was set also on the basis of the pull heuristics performance. We should point out that these situations where the control is taken away from the RL agent in fact are very unlikely if impossible to occur in practice due to the cost assignment scheme and the simulation model's parameters but there needs to be a way to handle them for consistency reasons. The complete state-action space then consists of 4(actions)×3^2(machine states) ×5^2(buffer states) ×5(bck states) = 4500 Q-values. As we have mentioned in former sections the only feedback available to the decision-maker is the reinforcement signal. It is essential that this signal has all the necessary information in order to communicate to the agent what we expect it to do. If we used Eq. 9 as the cost function, the agent would try to minimize the average WIP with no constraints whatsoever. The agent would minimize WIP easily simply by never giving the order to produce a single part. Our goal is to minimize WIP under the constraint of maintaining the service level above a specified target value. This is achieved by finding the appropriate values for parameters $q_i, i = 1,2$ and *c* from Equation 7. Parameters $q_i, i = 1,2$ can be interpreted as the cost of storing one part in output buffer *i* per time unit and *c* as the cost of unsatisfied demand per time unit. For example, if we increase parameter q_2, the resulting policy will yield lower WIP in buffer 2 at the expense of a lower service level. After a lengthy trial-and-error procedure we came up with $q_1 = q_2 = 10.0$ and $c = 100.0$ as the most suitable parameters for this task.

5.3 Results-Discussion

This section is devoted to the presentation and analysis of the numerical results that were produced via experimentation with simulation models. It is known from Reinforcement Learning theory that in order for the agent to converge to the true state-action values and average reward, all states and actions must be visited infinitely many times. Luckily, for most practical purposes a large amount of exploration will do the job, that way avoiding exhaustively long simulation times. Note that a key feature that renders RL superior to other intelligent methodologies is that it does not spend much time on evaluating state-actions with low probabilities of occurrence but focuses at maintaining reliable estimates for the Q-values of the more probable state-action pairs. In order to ensure convergence stability we set the decay factors of the learning rates to very low values. The RL algorithm was executed for 20 times the simulation time which was used to evaluate the performance of the pull type production control policies, that is, $20 \times 30000 = 600000$ time units. Given that the mean time between two successive customer demands was set to 1.66 time units (see section 5.1), approximately 360000 demands arrived to the system in this simulation time. All of these demands were ultimately satisfied by the system so there were, roughly, another 2(machines) × 360000(parts)=720000 "production in machine *i*" simulation events. If we take into consideration the events "failure in machine *i*" and "repair in machine *i*" the number of simulation events/decision making epochs is well beyond one million. This simulation time allows all state-action pairs to be visited many times and thus, calculate reliable Q-values estimates. Figure 1 shows the convergence of the RL algorithm. We set $Q_0(s,a) = 0.0, \forall s,a$. This implies that all admissible actions in all states have initially equal probabilities to be selected by the controller, including the optimal ones. As the number of iterations advances, the algorithm actually concentrates its choices to the perceived as optimal ones but this might not be clearly depicted in the average reward plot (Figure 1) due to the inaccuracy of the average reward estimate in the initial phase. Figure 1 illustrates the average reward estimate calculated by the RL algorithm and not the actual average reward. Note that in Singh's algorithm, the average reward estimate is calculated with the help of the immediate reward

and the state-action values $\max_{a'} Q_t(s',a')$ and $Q_t(s,a)$ (see Listing 1).

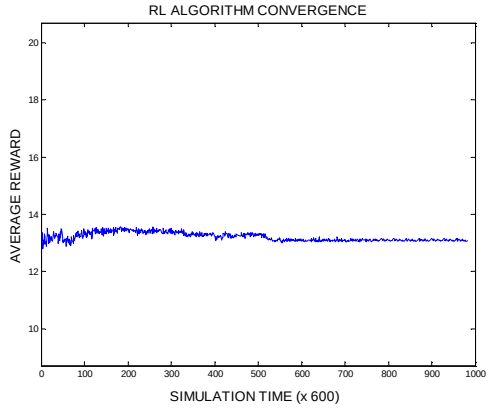


Figure 1. Average reward of the RL control policy

The initial heavy exploration phase is depicted in the fluctuations of the curve in Figure 1, while in the right-most part of the plot where the learning curve forms a straight line we can see the convergence phase of the algorithm. At this point we should mention that if the learning rates are prevented from decaying to zero the algorithm does not converge completely and is rendered suitable for a non-stationary problem where, i.e. the demand generating process slowly changes with time, a situation which is impossible to handle for the pull heuristics. The average WIP distribution through out the system and the backordered demands average together with the service rate that was achieved by the steady RL policy as well the ones of the three pull type control policies are presented in Table 2.

Table 2. Average buffers levels-backorders averages-service levels

	$\overline{O_1}$	$\overline{O_2}$	$\overline{Bck}$	*SL*
Kanban	0.39	2.88	0.18	93.12%
Base Stock	0.11	3.24	0.17	93.68%
CONWIP	0.05	3.28	0.17	93.70%
RL	0.09	3.11	0.18	93.10%

The simulated data are presented graphically in Figure 2. For all policies we reported service levels of approximately 93% and an average of about 0.18 unsatisfied customer orders. The characteristic attribute of the Kanban control policy which is tight coordination between the various manufacturing stations of the system is evident in the first row of Table 2. Indeed, the Kanban system displays the most uniform WIP distribution among all pull type policies but also it maintains the highest WIP inventory in buffer 1 during the system's operation. The Base Stock mechanism offers a significant improvement over the Kanban policy in terms of WIP accumulation in the first buffer thanks to the initially zero base stock in that buffer but this happens in the expense of WIP build-up in the finished goods storing facility. In a system that operates under the CONWIP policy all the machines except the first one have the perpetual authorization to produce whenever they can, and this is why WIP tends to accumulate in the last buffer.

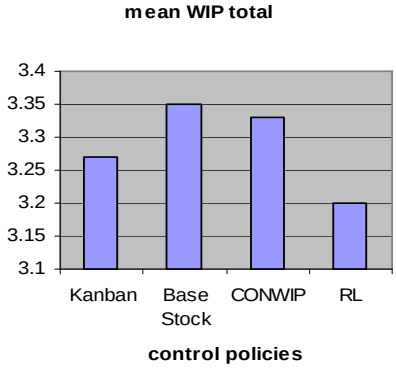


Figure 2. Total WIP

This situation is depicted in Table 2 where we observe that the average finished products inventory in the CONWIP system is the highest of all control policies. As a consequence, parts are not stalled in intermediate buffers due to e.g. lack of production authorization which in turn results in very low average WIP levels in the first buffer. The RL policy produced slightly higher WIP levels in the first buffer than the best policy found in this part which was CONWIP. It also ranked second in constraining WIP in the second buffer too. However the policy derived using the Reinforcement Learning algorithm outperforms the other three policies when it comes to total WIP as we can see in Figure 2. We could argue that the RL policy combines the desirable aspects of the various pull type control mechanisms while omitting their weaknesses. In order to gain insight of the behavior of the RL policy we examined the Q-values of state-action pairs that we were particularly interested in. It is obvious that the RL policy is considerably more complicated than the pull heuristics as it explicitly maps several hundreds of possible system states into actions and thus not as conceptually clear as these policies but this is not necessary a drawback in the modern automated industrial reality. The reasons why the RL policy outperforms the existing control policies can be sought in the fact that it is more state-aware than classical pull type mechanisms. None of the Kanban, Base Stock and CONWIP policies takes into consideration the events of machine break down and repair and this leads to sub-optimal behavior in many cases. For instance, consider the case where the second production facility is under repair. The CONWIP and Base Stock control policies will keep authorizing the first station to produce new parts regardless of the fact that these cannot be released to the downstream station thus, leading to excessive inventory accumulation in the first buffer. On the other hand, the Kanban control system will not authorize production in a station unless there is a free kanban authorization available, a condition that severely impedes the system's ability to react to customer arrivals. It is true that the RL policy may not behave optimally for numerous rarely visited states but the overall improvement offered to the system performance is more than evident.

6. CONCLUSIONS AND FUTURE RESEARCH

In this paper we use artificial intelligence methodologies to derive efficient, near-optimal control policies for serial manufacturing

lines. For this purpose we apply an algorithm from the field of Average Reward Reinforcement Learning. After a brief introduction to the central concepts in Reinforcement Learning and a description of the learning algorithm we proceed to a detailed presentation of the course of action that we followed in order to formulate the problem of production control as a Reinforcement Learning task. We evaluated the performance of the derived control policy through simulation and compared it with those of well-known pull type production control policies. Our experiments involved a manufacturing system with two unreliable machines in tandem where backordering is allowed and a stochastic demand generating process. The competing policies were evaluated on the basis of minimizing WIP subject to a service level constrain. The RL policy was found to outperform the Kanban, Base Stock and CONWIP production control policies in minimizing total WIP primarily due to the fact that it is more state-dependent than the above mentioned heuristics, as it is pointed out in the analysis accompanying the presentation of the numerical results. We plan to proceed to more extensive experimentation in the future in order to produce additional data as well as to extend the use of the RL methodology in manufacturing systems of a different type, as job-shops etc. The tabular approach that we use in this paper is somehow limiting in reference to the size of the problems that it can be used in. As a consequence, another logical direction for research would be to use some kind of function approximation, i.e. an artificial neural network in order to handle large scale problems

REFERENCES

[1] S. Axsäter and K. Rosling, 'Installation vs. echelon stock policies for multilevel inventory control', *Management Science*, **39(10)**, 1274-1279, (1993).

[2] R.E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.

[3] D. Bertsekas, *Dynamic Programming: deterministic and stochastic models*, Prentice Hall, New York, 1987.

[4] B.J. Berkley, 'A review of the kanban production control research literature', *Production and Operations Management*, **1(4)**, 393-411, (1992).

[5] J.A. Buzacott, J.G. Shanthikumar, *Stochastic Models of Manufacturing Systems*, Prentice Hall, New York, 1993

[6] T. Das, A. Gosavi, S. Mahadevan and N. Marchallek, 'Solving semi-markov decision problems using average reward reinforcement learning', *Management Science*, **45(4)**, 560-574, (1999).

[7] S.B. Gershwin, *Manufacturing Systems Engineering*, Prentice Hall, New York, 1994.

[8] A. Gosavi, 'A reinforcement learning algorithm based on policy iteration for average reward: empirical results with yield management and convergence analysis', *Machine Learning*, **55(1)**, 5-29, (2004).

[9] R. Howard, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge,1996.

[10] L.P. Kaelbling, M.L. Littman, A.W. Moore, 'Reinforcement Learning: a survey', *Journal of Artificial Intelligence Research*, **4**, 237-285, (1996).

[11] F. Karaesmen, Y. Dallery, 'A performance comparison of pull type control mechanisms for multi-stage manufacturing', *International Journal of Production Economics*, **68**, 59-71, (2000)

[12] G. Liberopoulos and Y. Dallery, 'A unified framework for pull control mechanisms in multi-stage manufacturing systems', *Annals of Operations Research*, **93**, 325-355, (2000)

[13] M.L. Littman, T.L. Dean, L.P. Kaelbling, 'On the complexity of solving MDPs', *In Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, Montreal, Quebec, Canada, (1995)

[14] S. Mahadevan, 'Average reward reinforcement learning: foundations, algorithms and empirical results', *Machine Learning*, **22**, 159-196, (1996).

[15] M.L. Putterman, *Markov Decision Processes*, Wiley Interscience, New York, 1994.

[16] A. Schwartz, 'A reinforcement learning method for maximizing undiscounted rewards', *In Proceedings of the Tenth Annual Conference on Machine Learning*, 298-305, (1993)

[17] S.P. Singh, 'Reinforcement learning algorithms for average-payoff markovian decision processes', *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, 202-207, (1994)

[18] M.L. Spearman, D.L. Woodruff, W.J. Hopp, 'CONWIP: a pull alternative to kanban', *International Journal of Production Research*, **28**, 879-894, (1990).

[19] Y. Sugimori, K. Kusunoki, F. Cho and S. Uchikawa 'Toyota production system and kanban system materialization of just-in-time and respect-for-humans systems', *International Journal of Production Research*, **15(6)**, 553-564, (1997).

[20] R.S. Sutton and A.G. Barto, *Reinforcement Learning: an introduction*, MIT Press, Cambridge, MA, 1998.

[21] P. Tadepalli and D. Ok, 'Model-based average reward reinforcement learning', *Artificial Intelligence*, **100**, 177-224, (1998).

[22] M.H. Veatch, L.M. Wein, 'Monotone control of queueing networks', *Queueing Systems*, **12**, 391-408, (1992).

[23] M.H. Veatch, L.M. Wein, 'Optimal control of a make-to-stock production system', *Operations Research*, **42**, 337-350, (1994).

[24] C.D. Paternina-Arboleda, T. K. Das, 'Inteligent dynamic control policies for serial production lines', *IIE Transactions*, **33**(1), 65-77, (2001)

[25] S. Mahadevan and G. Theocharous, 'Optimizing Production Manufacturing using Reinforcement Learning' , *In Proceedings of the Eleventh International FLAIRS conference* , 372-377, (1998).

1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2008)
Patras, Greece, July 22 2008

Efficient Learning of Dynamics Models Using Terrain Classification

Bethany R. Leffler and **Christopher R. Mansley** and **Michael L. Littman**[1]

Abstract. Terrain classification in robotics has heavily focused on determining a region for traversal, while also labeling obstacles. Our work attempts to expand this essentially binary viewpoint and to use terrain classifiers as an indicator for switching between a set of system dynamics. By learning multiple models of the system dynamics, the robot is able to assess alternative paths based on traversal costs of different terrain types instead of strict distance metrics. We demonstrate a system that reliably learns an optimal control policy using this additional terrain information and contrast it with several systems based on more traditional methods that fail to reliably complete the same task.

1 Introduction

Terrain classification for road following is often a binary road (passable) versus non-road (impassable) classification. As such, the robot cannot distinguish more or less passable terrains so as to prefer more passable terrains but allowing for less passable terrains to be traversed if alternatives are limited or if doing so would lead to a greatly reduced travel time. In this work, we take a more utility-oriented approach, distinguishing between a wider class of terrains, learning distinct dynamics models for each, and using these models to plan cheapest paths taking terrain into consideration.

The work in this paper frames this problem in the setting of reinforcement learning [16]. Recent work in the field has provided bounds on how much experience is needed to learn optimal control policies [10]. These bounds are typically proven with no assumptions about similarities between states, which leads to learning algorithms that scale at best linearly with the number of states. In a robotic domain, where the space of inputs is potentially infinite, these bounds are typically not useful. Instead, this paper builds on related work that exploits structure in the underlying state space to reduce the quantity of information needed to learn accurate models [12]. By using automatically extracted classes to index separate dynamics models for learning, the agent is able to perform more flexible path planning.

2 Related Work

A typical implicit assumption in robot-navigation research is that the robot has one dynamics model, which describes how it traverses from one state to the next. One example is in the representation of the state transition model in a Kalman filter's predict step, which captures the prediction of the next state from the current state [9]. One of the ways of ensuring that this assumption holds is to have the system follow surfaces appropriate for the model. This approach is often referred to as road following [5]. A common way to determine what is a road and what is not is to use a supervised learning algorithm to construct a terrain classifier [13]. When roads are not marked, such as on dirt roads or in clearings, a combination of sensors such as laser range finders and video cameras can be used in this binary terrain classification task to train an agent to learn to recognize the road through supervised [14] and self-supervised techniques [6]. Research has also been done in terrain modeling to determine the pass-ability of an area when the ground cannot be seen [17].

Figure 1. Image of the LEGO® Mindstorms NXT robot in the experimental environment.

These approaches, when successful, result in the robot navigating correctly to its goal location along a single terrain class. However, the path taken might not be optimal in a utility-theoretic sense. By using the assumption that there is only one dynamics model in the world, the agent is unable to fully calculate the best path to the goal. The plan or policy generated would be sub-optimal because a single model would try to encompass both the good (road) and bad (off-road) parts of the world; this oversimplification can lead to improper policy cost estimation. By learning multiple dynamics models, the agent can more fully model the dynamics of the environment and calculate a better policy.

From a reinforcement-learning perspective, navigation algorithms are often assessed based on the agent's learned policy and the how far that policy is from optimal. Obtaining a good policy often means fully exploring the environment. Exploration, though, comes at a cost, and therefore must be done in an efficient manner [1]. Even an efficient exploration algorithm may not be enough to converge to a good policy in a reasonable amount of time. For this reason, many algorithms use generalization techniques, such as function approximation, to limit the amount of exploration needed to learn about the entire environment [8].

[1] Rutgers University

Our contributions in this paper include using classes or types extracted from images of terrain to allow for the generalization of action models across states, which speeds up the learning time while allowing for efficient exploration. We also demonstrate experimental environments that cannot be completed reliably without the use of separate state dynamics, showing that not only does this technique speed learning, but some situations actually require it.

3 Background

Previously, we introduced the idea of relocatable action models (RAM) to enhance exploration [11]. Instead of using the traditional Markov decision process (MDP) representation of states, actions, and transition functions, we used the RAM representation of states, actions, *types* and *outcomes* [15]. This alternative representation of the underlying MDP is defined by 8-tuple $\langle S, A, C, O, \kappa, t, \eta, r \rangle$, where S is a set of states, A is a set of actions, C is a set of *types*, O is a set of *outcomes*, $r : S \rightarrow \Re$ is the state dependent reward function, and $\kappa : S \rightarrow C$ is a mapping indicating how each state maps to a type or cluster. The function $t : C \times A \rightarrow \Pr(O)$ is a mapping known as the *relocatable action model*. It captures effects of different actions in a state-independent way by mapping a type and an action to a probability distribution over possible outcomes. In this work, outcomes are the change in location and orientation in robot coordinates. The mapping $\eta : S \times O \rightarrow S$ is the *next-state function*. It takes a state and an outcome and computes the resulting next state by transforming from robot coordinates to world coordinates.

In this paper, our vision-based terrain-classification system is what defines the type mapping κ. This mapping converts perceptual features of the state space into a finite set of terrain types. We show that κ can be computed *a priori* using a generic vision system. With κ and η defined, the learner can focus on the terrain-specific action model, resulting in fast, accurate learning.

Whereas prior work [11] used a hand-tuned classification of states to types, the current paper shows that this mapping can be extracted automatically and used successfully in the learning setting.

4 System Architecture

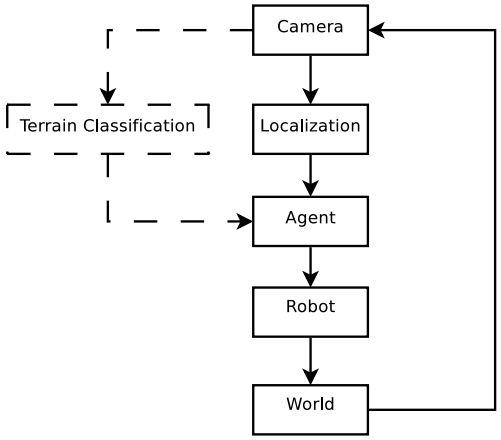


Figure 2. Flow chart of the system architecture. The dashed lines indicate information passing that occurs at startup.

Figure 2 shows the flow of data through our system for a robotic domain. Before the robot is placed in the environment, a picture is taken with an overhead camera and sent through an image-segmentation engine to determine terrain classification (see Section 4.1). Classification information is then stored as an additional feature for each state.

Once the state space is defined, the robot is placed in its starting configuration and the agent queries the localization system for the robot's position in the world. Using this information, the agent, with the guidance of the RAM-Rmax algorithm (see Figure 3), chooses which action to take based on the outcomes it has previously seen.

The selected action is then sent to the robot to execute. After execution is complete, the agent once again retrieves the robot's location information and uses it to calculate the latest outcome, which is added to a list of outcomes seen in the same terrain type. The agent then chooses the next action to take. This process continues until the localization system tells the agent that the robot is in the goal region or out of bounds. These occurrences end an episode; the robot is placed back in the starting location to execute another episode.

4.1 Terrain Classification

We used a IEEE1394 video camera to take an image of the world without the robot and then fed the image into the Edge Detection and Image SegmentatiON (EDISON) system [4] where similar terrains are determined based on color, texture, and proximity of pixels. Since the image-segmentation system determines the number of clusters on its own, the one parameter to be set is the minimum number of pixels that a cluster can contain. This value is set to the number of pixels that the robot occupies in the image to ensure that all of the robot's wheels can occupy a single patch of terrain at the same time.

The clustered image that EDISON returns is then converted into an indexed image. To limit the number of spurious clusters, all clusters that appear in fewer than $\frac{|S|}{10}$ states have their pixels reassigned to adjacent segments. The remaining clusters are used to determine the terrain-type feature of the state space. These terrain classifications will be used to determine to which dynamics model the learned outcomes will be applied.

4.2 Localization

The localization system is a standard fiducial-based system, which for these experiments acts as an indoor global positioning system (GPS). Using the same overhead camera as above, the location of a marker affixed to the robot is obtained using commonly available color-segmentation software [2]. The type of marker used is based on work done for a robotic soccer application [3]. The typical accuracy of this system is less than 5mm.

4.3 Learning The Dynamics Model

After an action is taken in state s and the position of the new state s' is fed back to the agent, the displacement and change in orientation between s and s' are calculated. These differences are stored as a list of *outcomes* for $\kappa(s)$. Instead of maintaining a list with an outcome for every action taken, which could grow without bound, the list of outcomes keeps a tally t_C of how many times that outcome C has been seen. This approach allows for proper calculation of the probability of seeing that outcome while minimizing the size of the list that the algorithm has to traverse. Since the number of possible outcomes is finite due to the granularity of the localization system, the maximum size of the outcome list is also finite.

4.4 Planning

When deciding which actions to take in the environment, the agent uses a parameterized variation of the Rmax algorithm that sets the expected reward of taking an action in a terrain to the maximum reward, R_{max}, if that combination of terrain and action has not been experienced often. An action becomes "known" for a particular terrain when the robot has performed that action in that terrain M times, where M is a free parameter. Once the action is "known", the value for that action becomes the solution to $Q(s,a) =$

$$r(s,a) + \gamma \left(\sum_{o \in O} \frac{t_C(\kappa(s),a,o)}{z} \times \max_{a' \in A} Q(\eta(s,o),a') \right), \quad (1)$$

which is updated as t_C changes Here, z is a normalization constant.

To calculate values, all outcomes for that terrain type need to be mapped back from feature space into state space. For instance, if the robot is in state s with $x = 35$, $y = 25$, $\theta = 90.0$ and the outcome o_1 is a displacement of magnitude 2.0 in direction 315.0 and the change in orientation is 314.0 degrees, then s' for that outcome would be the state with the features $x = 36$, $y = 26$, $\theta = 44.0$. This calculation is done for every outcome and the reward for each possible s' is weighted by its probability and summed.

Once the value for each action is calculated by the algorithm (see Figure 3), the agent chooses the action with the highest value and sends the corresponding command to the robot via Bluetooth®. While planning can take several milliseconds, these calculations can be performed while the robot is performing its actions (each action takes approximately one second), resulting in no computational delay.

Global data structures: Q, t_C
Constants: R_{max}, M

INITIALIZE():
  for $c \in C, a \in A, o \in O$:
    $t_C(s,a,o) = 0$
  for $s \in S, a \in A$:
    $Q(s,a) = R_{max}$

UPDATE(s, a, s'):
  $o = s' - s^{\dagger}$
  $t_C(\kappa(s),a,o) = t_C(\kappa(s),a,o) + 1$
  for $s \in S$:
    for $a \in A$
      $Q(s,a) = R_{max}$
  repeat until $Q(s,a)$ stops changing:
    for $s \in S$:
      for $a \in A$:
        $z = \sum_{o \in O} t_C(\kappa(s),a,o)$
        if $z \geq M$
          $Q(s,a) = r(s,a) + \gamma \sum_{o \in O} \frac{t_C(\kappa(s),a,o)}{z} \times \max_{a' \in A} Q(\eta(s,o),a')$

† This subtraction is a transformation expressing s' in the coordinate frame of s.

Figure 3. The RAM-Rmax algorithm.

5 Experiment

To quantify the benefits of using image segmentation for better performance, experiments were performed in a real world robotic environment.

5.1 Experimental Setup

For our experiment, we ran a LEGO® Mindstorms NXT (see Figure 1) on a multi-surface environment. This domain, shown in Figure 4, consisted of a highly variable region comprised of rocks embedded in wax and a more deterministic carpeted area. The goal was for the agent to begin in the start location (indicated in the figure by an arrow) and end in the goal without going outside the environmental boundaries. The rewards were -1 for going out of bounds, $+1$ for reaching the goal, and -0.01 for taking an action. Reaching the goal and going out of bounds ended the episode and resulted in the agent getting moved back to the start location.

One difficulty of this environment is the difference in dynamics models. Figure 5 shows the outcomes observed by the robot on both the rock and carpet surfaces. The center of the circle represents the starting location of the robot. The dashed lines indicate the angle (in degrees) and distance (in pixels) of displacement. From left to right, this figure shows the outcomes of the left turn, go forward, and right turn actions on for the rock (top) and carpet (middle) surfaces. The bottom row shows the same outcomes as above, but combines the terrains to demonstrate the amount of noise that is introduced when the terrains are not distinguished. Some actions, such as turning right on rocks, are more sparse than others due to the number of times that an action was taken during exploration.

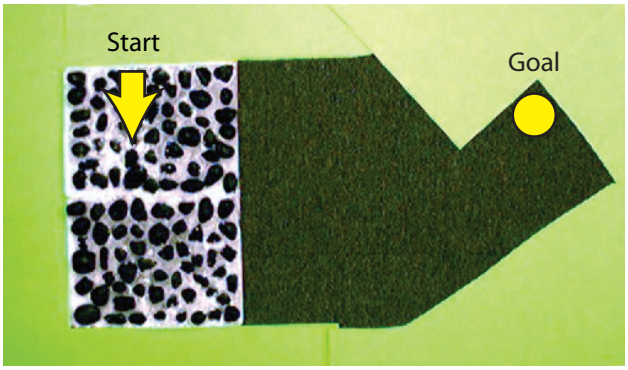
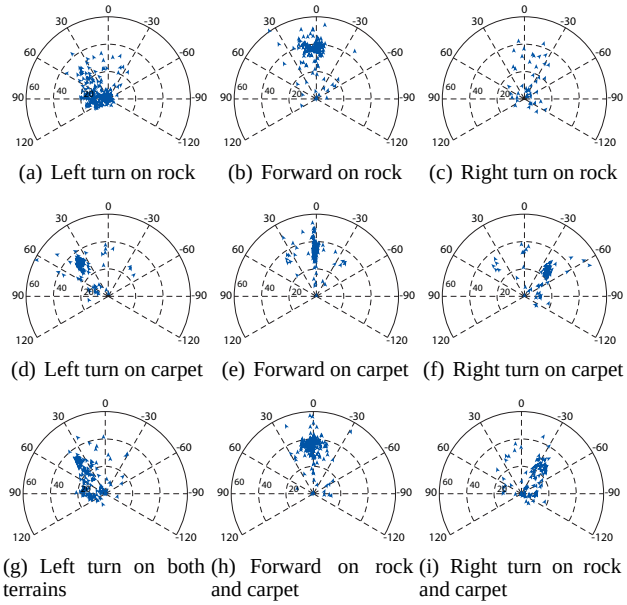


Figure 4. Image of the environment. The start location and orientation is marked with an arrow. The goal location is indicated by the circle. Green pieces of poster board are shown here marking the boundaries of the environment.



(a) Left turn on rock (b) Forward on rock (c) Right turn on rock

(d) Left turn on carpet (e) Forward on carpet (f) Right turn on carpet

(g) Left turn on both terrains (h) Forward on rock and carpet (i) Right turn on rock and carpet

Figure 5. Outcomes learned by the robot for different actions and surfaces.

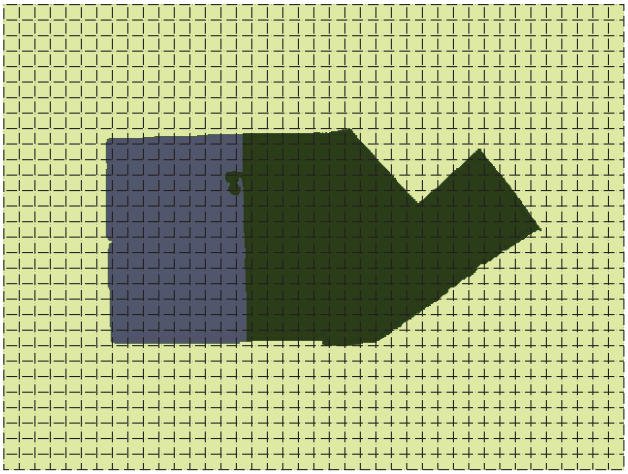


Figure 6. Resulting discretized segmented image from EDISON of the environment showing two different surface types. Several states were mislabeled due to the image processing algorithm, but these mislabelings did not harm the results.

Due to the close proximity of the goal to boundary, the agent needs an accurate dynamics model to reliably reach the goal. To make this task even more difficult, the actions were limited to going forward, turning left, and turning right. Not allowing the agent to move backwards increased the need for the agent to accurately approach the goal reliably. For example, if the robot enters the narrow portion of the environment facing away from the goal, it is not able to turn around without going out of bounds. As such, a robot with an inaccurate transition model would be likely to think this task is impossible.

For the experiments, we compared the RAM-Rmax and fitted Q-learning [7] algorithms with and without image segmentation. We would have liked to run the Rmax algorithm for a comparison, but it was not plausible to do so due to the robot's battery life—too much experience was needed to train the algorithm. All algorithms were informed of the reward function—it did not need to be learned. The algorithm with no image segmentation learns one action model for the entire domain. Figure 6 shows the results of the EDISON image-segmentation engine when fed in the image of the world and the minimum region to segment. Recall that the minimum region to segment was specified to be the number of pixels that the robot occupies in the image, which for this experiment was 4000 pixels.

For both agents, the world was discretized to a forty by thirty by ten state space instead of the camera's full resolution of 640 by 480 by 360 degrees of orientation. This coarse discretization was used to limit the number of states that the robot could occupy at once. Lastly, each algorithm had the value of M set to ten, which was determined after informal experimentation.

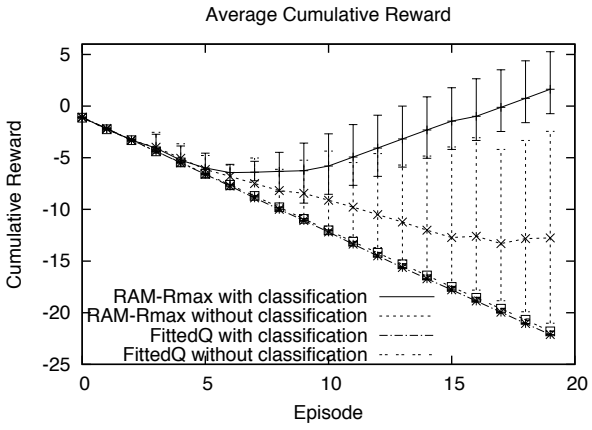


Figure 7. Graph of the RAM-Rmax and fitted Q-learning algorithms' average cumulative reward with and without terrain classification via image segmentation.

6 Results

Figure 7 shows the average performance and standard deviation of the RAM-Rmax and fitted Q-learning algorithms with and without image segmentation over five runs of twenty episodes. When RAM-Rmax used image segmentation to determine the number of surface types in the environment, RAM-Rmax reached the goal in 61% of

the episodes as opposed to 22% of the episodes when using the assumption that all surfaces were the same. This difference is shown in greater detail when looking specifically in the last 10 episodes after some learning had taken place. Narrowed to these instances, the success rates are 96% and 34%, respectively. Fitted Q-learning was not able to reach the goal in any of the runs with or without the image segmentation. Doubling the number of episodes to 40 in a run also did not result in any positive reward. Indeed, published results with this algorithm suggest hundreds or thousands of episodes are often needed.

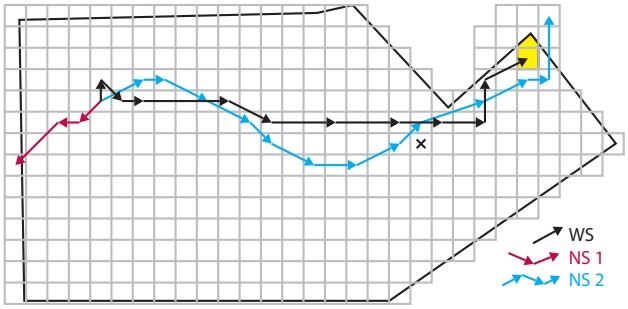


Figure 8. Diagram showing paths learned by the algorithm with segmentation (WS) and with no segmentation (NS) en route to the goal (shown in yellow). NS1 demonstrates a sample path in which the agent judged the goal as unreachable and so minimized negative reward by exiting the environment quickly. NS2 shows a sample path in which the agent's inaccurate model causeed it to miss the goal. "X" marks the state used for the example in Section 7.

The figure also shows a great variation in the performance of the RAM-Rmax algorithm when it did not use image segmentation. The reason for this difference is the variability in the dynamics model that the agent learned for each run. By learning varying dynamics models, the chosen path of the same agent changed drastically between runs as shown Figure 8. In two of the five runs, the agent thought that it was valuable to navigate towards the goal from early in the run. However, the dynamics model learned was noisy, and the agent would accidentally drive out of bounds when approaching the goal. In the other half of the runs, the agent did not think that it was possible to reach the goal based on its learned dynamics model, and, therefore, would drive out of the environment as quickly as possible to minimize negative reward.

In contrast, the agent that used image segmentation learned that the rocky surface was unpredictable, but that the carpet surface allowed for consistent actions. Once these two surfaces were learned, the agent was able to arrive at the goal reliably, seldom over-shooting of the goal, as shown in Figure 8.

7 Discussion

The two RAM-Rmax algorithms outperformed both of the fitted Q-learning algorithms due to the efficiency with which they use experience data. RAM-Rmax with and without image segmentation was able to use its learned model to generalize between states because of the similarity of the dynamics in these states. Since fitted Q-learning does not model the environment, its generalization ability was limited to exploiting local consistency in the value function. Throughout the twenty episodes, neither of the fitted Q-learning algorithms had been able to take advantage of any structure and were still active exploring to learn values. As such, they had learned very little and ended up going out of bounds every time.

The performance discrepancy of the two RAM-Rmax algorithms can be better explained when looking at the learned value function of the two algorithms. For example, when using the RAM-Rmax algorithm with image segmentation, the average expected reward of a state near the goal ($X = 25$, $Y = 15$, $\theta = 0$, marked with an "X" in Figure 8) was 0.450 with a standard deviation of 0.194 in comparison with the algorithm without image segmentation which on average calculated the value of the same state to be 0.1782 with a standard deviation of 0.373. These values vary because of the dynamics models learned. The algorithm without image segmentation performed as if the goal was on the rocky surface. By increasing the amount of context the RAM-Rmax algorithm uses to distinguish the dynamics, it is able to better model its environment.

However, there is a limit to how much improvement additional context can add. If the terrain classifier were to have found three types instead of two, the agent might have been able to model the dynamics of when its front wheels were on one surface and its back wheel on another. If the agent declared each rock its own surface, the agent would be able to model its likelihood of getting stuck on that rock. The problem with this approach is scalability. The more surface types the learner recognizes, the less it generalizes and the more exploration it needs to do. In the limit, as the number of types approaches the number of states, this algorithm becomes equivalent to (non-generalizing) Rmax.

By scaling the number of terrain types, an algorithm implicitly assumes information about the structure of the environment. At one end of the scale, the assumption is that there is one class that all the world adheres to as in road following. At the other, each state has its own idiosyncrasies and, therefore, should be modeled individually.

8 Conclusions and Future Work

Optimal decisions for path planning require accurate models for predicting the outcomes of actions. In this paper, we examined a method for building accurate models by partitioning learning experiences according to a set of automatically extracted terrain classes. We found that the resulting dynamics models led to a utility-based path planning system that learned quickly to make effective decisions. Compared to approaches that overgeneralize (estimate a single transition model) or undergeneralize (estimate separate models for each state), this approach makes effective use of experience and succeeded at reaching the goal location at a significantly higher rate in our experiments.

Currently, the system is dependent on the sensory features correlating well with differences between terrains. In future work, we will explore using feature selection to determine which of the robot's multiple sensors best predict terrain characteristics. By doing so, the agent could learn when to use, for example, ladar, IR, and satellite imagery to best estimate terrain features that matter for predicting action outcomes.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grants ITR IIS-0325281 and DGE 0549115.

REFERENCES

[1] Ronen I. Brafman and Moshe Tennenholtz, 'R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning', *Journal of Machine Learning Research*, **3**, 213–231, (2002).
[2] James Bruce, Tucker Balch, and Manuela Veloso, 'Fast and inexpensive color image segmentation for interactive robots', in *Proceedings of IROS-2000*, Japan, (October 2000).
[3] James Bruce and Manuela Veloso, 'Fast and accurate vision-based pattern detection and identification', in *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, (May 2003).
[4] Christopher M. Christoudias, Bogdan Georgescu, and Peter Meer, 'Synergism in low level vision', in *ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 4*, p. 40150, Washington, DC, USA, (2002). IEEE Computer Society.
[5] J. Crisman and Chuck Thorpe, 'UNSCARF, a color vision system for the detection of unstructured roads', in *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pp. 2496–2501, (April 1991).
[6] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski, 'Self-supervised monocular road detection in desert terrain', in *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, (August 2006).
[7] Damien Ernst, Pierre Geurts, and Louis Wehenkel, 'Tree-based batch mode reinforcement learning', *J. Mach. Learn. Res.*, **6**, 503–556, (2005).
[8] Andrew Moore Justin Boyan, 'Generalization in reinforcement learning: Safely approximating the value function', in *Neural Information Processing Systems 7*, eds., G. Tesauro, D.S. Touretzky, and T.K. Lee, pp. 369–376, Cambridge, MA, (1995). The MIT Press.
[9] Rudolph Emil Kalman, 'A new approach to linear filtering and prediction problems', *Transactions of the ASME–Journal of Basic Engineering*, **82**(Series D), 35–45, (1960).
[10] Michael J. Keans and Satinder P. Singh, 'Near-optimal reinforcement learning in polynomial time', *Machine Learning*, **49**(2–3), 209–232, (2002).
[11] Bethany R. Leffler, Michael L. Littman, and Timothy Edmunds, 'Efficient reinforcement learning with relocatable action models', in *AAAI-07: Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pp. 572–577, Menlo Park, CA, USA, (2007). The AAAI Press.
[12] Bethany R. Leffler, Michael L. Littman, Alexander L. Strehl, and Thomas J. Walsh, 'Efficient exploration with latent structure', in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, (June 2005).
[13] Dean A. Pomerleau, 'ALVINN: An autonomous land vehicle in a neural network', 305–313, (1989).
[14] Christopher Rasmussen, 'Combining laser range, color, and texture cues for autonomous road following', in *IEEE International Conference on Robotics and Automation*, (2002).
[15] Alexander A. Sherstov and Peter Stone, 'Improving action selection in MDP's via knowledge transfer', in *Proceedings of the Twentieth National Conference on Artificial Intelligence*, (July 2005).
[16] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.
[17] Carl Wellington, Aaron Courville, and Anthony (Tony) Stentz, 'Interacting Markov random fields for simultaneous terrain modeling and obstacle detection', in *Proceedings of Robotics: Science and Systems*, (June 2005).