J. Intelligent Learning Systems & Applications, **2009, 1, 28-41**
Published Online November 2009 in SciRes (www.SciRP.org/journal/jilsa)

Control Task for Reinforcement Learning with Known Optimal Solution for Discrete and Continuous Actions

Michael C. RÖTTGER[1], Andreas W. LIEHR[2]

Service-Group Scientific Information Processing, Freiburg Materials Research Center, Freiburg, Germany.
Email: [1]roettm@gmx.de, [2]Liehr@fmf.uni-freiburg.de

Received November 10th; 2008; revised January 29th, 2009; accepted January 30th, 2009.

ABSTRACT

The overall research in Reinforcement Learning (RL) concentrates on discrete sets of actions, but for certain real-world problems it is important to have methods which are able to find good strategies using actions drawn from continuous sets. This paper describes a simple control task called direction finder *and its known optimal solution for both discrete and continuous actions. It allows for comparison of RL solution methods based on their value functions. In order to solve the control task for continuous actions, a simple idea for generalising them by means of feature vectors is presented. The resulting algorithm is applied using different choices of feature calculations. For comparing their performance a simple measure is introduced.*

***Keywords*:** *comparison, continuous actions, example problem, reinforcement learning, performance*

1. Introduction

In Reinforcement Learning (RL), one solves optimal control problems without knowledge of the underlying system's dynamics on the basis of the following perspective: An agent which is aware of the current state of its environment, decides in favour of a particular action. The performance of the action results in a change of the agent's environment. The agent notices the new state, receives a reward, and decides again. This process is repeated over and over and may be terminated by reaching a terminal state. In the course of time the agent learns from its experience by developing a strategy which maximises its estimated total reward.

There are various RL methods for searching optimal policies. For testing, many authors refer to standard control tasks in order to enable comparability; others define their own test-beds or use a specific real-world problem they want to solve. Popular control tasks used for reference are e.g. the *mountain car problem* [1,2,3], the *inverted pendulum problem* [4], the *cart-pole swing-up task* [5,6,7] and the acrobot [2,8,9].

In most papers, algorithms are compared by their effectiveness in terms of maximum total rewards of the policies found or the number of learning cycles, e.g. episodes, needed to find a policy with an acceptable quality. Sometimes the complexity of the algorithms is also taken into account, because a fast or simple method is more likely to be usable in practice than a very good but slow or complicated one. All these criteria allow for some kind of *black-box testing* and can give a good estimation whether an algorithm is suitable for a certain task compared to other RL approaches. However, they are not very good at giving a hint where to search for causes why an algorithm cannot compete with others or why, contrary to expectations, no solution is found. The possible reasons for such kinds of failure are manifold, starting from bad concepts to erroneous implementations.

Many ideas in RL are based on representing the agent's knowledge by value functions, so their comparison among several runs with different calculation schemes or with known optimal solutions gives more insight into the learning process and supports systematic improvements of concepts or implementations.

One control task suitable for this purpose is the *double integrator* (e.g. [10]), which has an analytically known optimal solution for a bounded continuous control variable (the applied force) and the corresponding state-value function $V(s)$ can be calculated. Since this produces a bang-bang-like result, the solution for a continuous action variable does not differ from a discrete-valued control.

In this paper, we present a simple control task, which

has a known optimal solution for both discrete and continuous actions, with the latter allowing for better cumulative rewards. This control task is a simplified variant of the *robot path finding problem* [11,12]. Since its state is described by real-valued numbers, it is useful to calibrate new algorithms working on continuous state spaces before tackling more difficult control tasks in order to rule out the teething problems. It allows for a detailed comparison of RL solutions in terms of the state-value function V and action-value function Q and may serve for didactic purposes when discussing solution methods for discrete and continuous actions.

In real-world problems, the state space is often uncountable and infinitely large, e.g. when the state is described by real numbers. For this kind of tasks, simple tables cannot be used as memory for saving all possible values of $V(s)$ and $Q(s, a)$. A common way to handle this problem is the usage of function approximation, which introduces some inaccuracy to the system. Knowing an exact solution may help to develop heuristics to find appropriate parameters approximating the optimal solution well.

The paper is structured as follows: Section 2 introduces the notation describing RL problems and their solutions. The proposed control task and its optimal solution are presented in Section 3. To solve this problem for discrete and continuous actions by means of RL we applied a well-known algorithm combined with feature vectors for actions. This is discussed in Section 4. In Section 5 we present simulation results for discrete and continuous actions. Section 6 concludes the paper with a summary and an outlook.

2. Notation

In terms of RL, a control task is defined for a specific environment, in which an agent has to reach an objective by interacting with the environment. The learning process is organised in one or more episodes. An episode starts with an initial state s_0. It can either last forever (*continuing task*) or end up with a *terminal* or *absorbing state* S_T after performing T actions (*episodic task*). At time t, the agent becomes aware of the current state s_t of the environment and decides in favour of an action a_t. Performing this action results in a new state s_{t+1} and the agent receives a reward r_{t+1}. For an episodic task, the sequence of states, rewards and actions is denoted as

$$s_0 \overset{a_0}{\rightarrow} (r_1, s_1) \overset{a_1}{\rightarrow} (r_2, s_2) \overset{a_2}{\rightarrow} \ldots \overset{a_{T-1}}{\rightarrow} (r_T, s_T).$$

Figure 1 shows the information exchange between the agent and its environment.

A deterministic *policy* π is a mapping from states $s \in \mathbb{S}$ to actions $a \in \mathbb{A}$:

$$\pi : \mathbb{S} \rightarrow \mathbb{A}.$$

In general, for a given state s, the action space $\mathbb{A}(s)$ also depends on the state s, because in some states some actions may not be available. Here, the state should be representable as a tuple of n real numbers, so $\mathbb{S} \subset \mathbb{R}^n$. The action space is considered in two variants: A finite set $\mathcal{A} = \{\overline{a}_1, \ldots, \overline{a}_m\}$, whose elements will be denoted as *discrete actions*, and alternatively an infinite set $\mathbb{A} \subset \mathbb{R}^m$, whose elements are called *continuous actions*. For the control task presented in Section 3, both cases are discussed. In general and for real-world control tasks, also mixed action spaces having both discrete and continuous actions are imaginable.

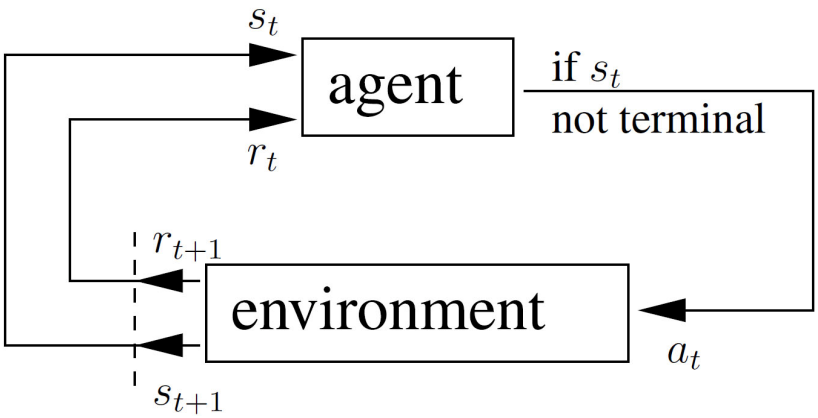


Figure 1. RL's perspective of control tasks as described in [13]. The diagram shows the communication of the agent with the environment.

In order to handle episodic and continuing tasks with the same definitions for value functions, we follow the notation of [13]: Episodic tasks with terminal state s_T can be treated as continuing tasks remaining at the absorbing state without any additional reward: $s_{T+k} \equiv s_T$ and $r_{T+k} \equiv 0$ for $k = \{1,2,3,\ldots\}$.

Using this notion, for all states $s \in \mathbb{S}$, the *state-value function*

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1}\middle| s_t = s\right\} \quad (1)$$

concerning a given policy π holds the information, which total discounted reward can be expected when starting in state s and following the policy π. The quantity $\gamma \in [0,1]$ is called *discount factor*. For $\gamma < 1$ the agent is myopic to a certain degree, which is useful to rank earlier rewards higher than latter ones or to limit the cumulative reward if $T = \infty$, i.e. no absorbing state is reached.

In analogy to the value of a given state, one can also assign a value to a pair (s, a) of state $s \in \mathbb{S}$ and action $a \in \mathbb{A}(s)$. The *action-value function*

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1}\middle| s_t = s, a_t = a\right\} \quad (2)$$

concerning a policy π is the estimated discounted cumulative reward of an agent in state s, which decides to perform the action a and then follows π.

For the deterministic episodic control task proposed in this paper we have chosen $\gamma = 1$, so the Equations (1) and (2) simply reduce to finite sums of rewards.

In order to maximise the agent's estimated total reward, it is useful to define an ordering relation for policies: A policy π is "better than" a policy π', if

$$\pi \geq \pi' :\Leftrightarrow V^{\pi}(s) \geq V^{\pi'}(s)\ \forall s \in \mathbb{S}. \quad (3)$$

An *optimal* policy complies with

$$\pi^{*} \geq \pi \quad \forall \pi \quad (4)$$

and is associated with the *optimal value functions* V^{*} and Q^{*}, which are unique for all possible optimal policies:

$$Q^{*}(s,a) := Q^{\pi^{*}}(s,a), \quad (5)$$

$$V^{*}(s) := V^{\pi^{*}}(s) = \max_{a} Q^{*}(s,a). \quad (6)$$

Given the optimal action value function Q^{*} and a current state s, an optimal action a^{*} can be found by evaluating

$$a^{*}(s) = \arg\max_{a} Q^{*}(s,a). \quad (7)$$

3. The Directing Problem and Its Solution

3.1 The Control Task

The directing problem is a simplified version of the path finding problem [11,12] without any obstacles. The agent starts somewhere in a rectangular box given by

$$\mathbb{S} = \{(x,y) \mid 0 \leq x \leq x_{\max} \wedge 0 \leq y \leq y_{\max}\}. \quad (8)$$

Its real-valued position $s=(x, y)$ is the current state of the system.

The objective of the agent is to enter a small rectangular area in the middle of the box (Figure 2). This target area is a square with centre (c_x,c_y) and a fixed side length of $2d_T$. Each position in the target area, including its boundary, corresponds to a terminal state.

In order to reach this target area, the agent takes one or more steps with a constant step size of one. Before each step, it has to decide, which direction φ to take. If the resulting position is outside the box, the state will not be changed at all. So given the state $s = (x, y)$, the action $a = \varphi$, and using the abbreviation

$$(\tilde{x},\tilde{y}) = (x + \cos\varphi, y + \sin\varphi), \quad (9)$$

the next state $s' = (x', y')$ is

$$(x',y') = \begin{cases} (\tilde{x},\tilde{y}) & \text{if } (0 \leq \tilde{x} \leq x_{\max} \wedge 0 \leq \tilde{y} \leq y_{\max}), \\ (x,y) & \text{otherwise.} \end{cases} \quad (10)$$

In terms of RL, the choice of φ is the agent's action which is always rewarded with $r \equiv -1$ regardless of the resulting position. By setting the discount factor $\gamma = 1$, the total reward is equal to the negative number of steps needed to reach the target area.

For the analytical solution and for the simulation results, $x_{\max} = y_{\max} = 5$, $d_T = \frac{1}{2}$ and $c_x = c_y = 2.5$ have been chosen.

The optimal solution for this problem, although it is obvious to a human decision maker, has two special features:

- In general, the usage of continuous angles allows for larger total rewards than using only a subset of discrete angles and

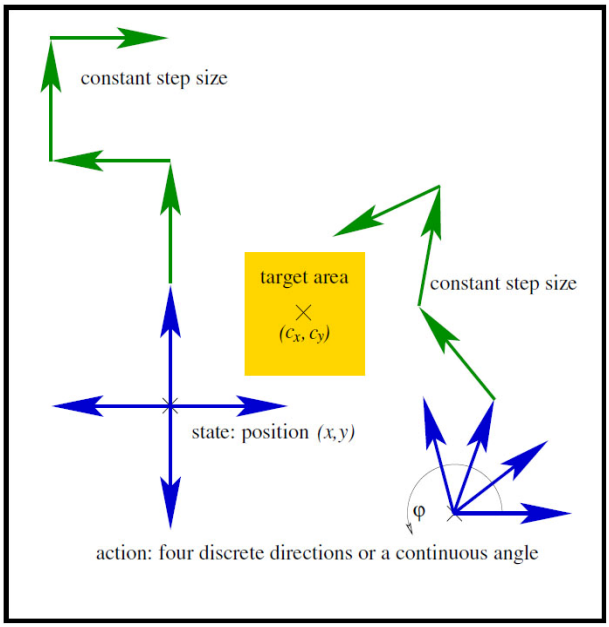


Figure 2. The direction finder. The state of the agent is its real-valued position somewhere in the simulation domain. Its objective is to enter the shaded target area by choosing a direction before each constant-sized step. This choice may be completely free (*continuous actions*) or limited to a finite number of four angles (*discrete actions*). The reward after each step is -1, so an optimal policy would minimise the total number of steps.

- the state value function V^* and Q^* can be expressed analytically, so intermediate results of reinforcement learning algorithms can be compared to the optimal solution with arbitrary precision.

In the following, the solutions for discrete and continuous actions are described.

3.2 Discrete Actions

For a finite set

$$\mathcal{A} = \{\rightarrow, \uparrow, \leftarrow, \downarrow\} \tag{11}$$

of four discrete directions, which can be coded as $\left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$ using angles in radians, the state value for an optimal policy for an arbitrary position (x, y) is given by

$$V^*_{\text{discr}}(x,y) = -\lceil h(g_x(x)) \rceil - \lceil h(g_y(y)) \rceil. \tag{12}$$

The symbol $\lceil \cdot \rceil$ denotes the *ceiling* function, which returns the smallest integer equal to or greater than a given number, and the terms $h(g_x(x))$ and $h(g_y(y))$ are abbreviations for

$$g_x(x) = | x - c_x | - d_T \tag{13}$$

$$g_y(y) = | y - c_y | - d_T \tag{14}$$

$$h(z) = \begin{cases} 0 & \text{for} \quad z \le 0 \\ z & \text{for} \quad z > 0 \end{cases} \tag{15}$$

The value of a terminal state is zero, since no step is needed.

Figure 3 shows a plot of the optimal state-value function for discrete actions. Due to the limited set of directions, the agent needs four steps when starting in a corner of the domain.

3.3 Continuous Actions

If arbitrary angles $\varphi \in [0, 2\pi]$ are allowed, i.e. continuous actions, the optimal state value is given by:

$$V^*_{\text{cont}}(x,y) = -\left\lceil \sqrt{h^2(g_x(x)) + h^2(g_y(y))} \right\rceil. \tag{16}$$

As for discrete actions, the helper functions defined in (13), (14), and (15) have been used.

Figure 4 shows a plot of the optimal state-value function for continuous actions. Contrary to the case of discrete actions, the agent only needs three steps at the most to enter the target area. Except for the set of possible directions, the task specification is the same as for discrete actions.

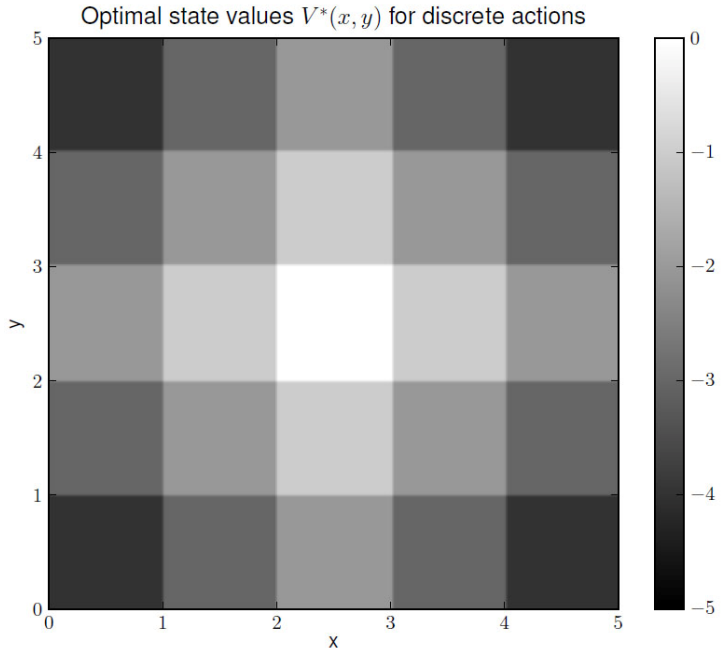


Figure 3. Optimal state value function $V^*(x, y)$ for four discrete actions → , ↑ , ← and ↓ . Because of $\gamma = 1$ it is equal to the negative number of steps needed in order to enter the target square in the centre, so $V^*(x, y) \in \{-4,-3,-2,-1,0\}$. $x_{max} = y_{max} = 5$, $d_T = \frac{1}{2}$ and $c_x = c_y = 2.5$.

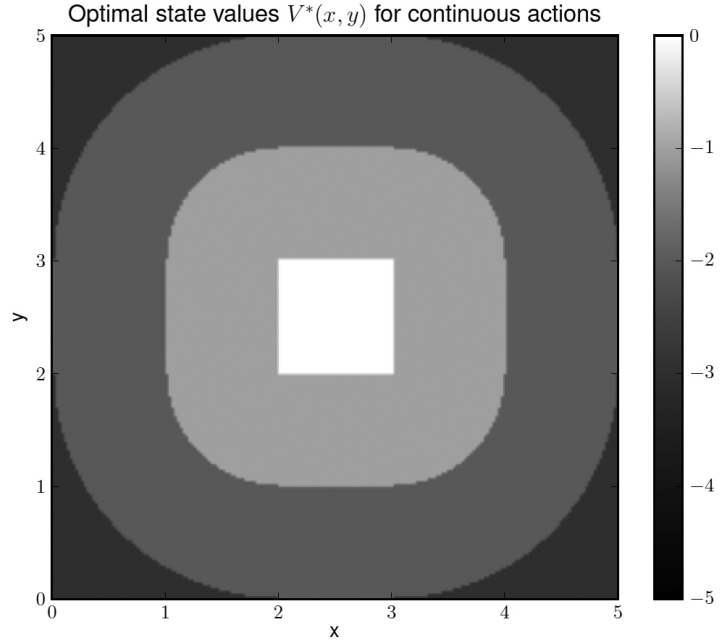


Figure 4. Optimal state value function $V^*(x, y)$ for continuous actions, i.e. $\varphi \in [0, 2\pi[$. The state value is equal to the negative number of steps needed to enter the target square in the centre, so $V^*(x, y) \in \{-3,-2,-1,0\}$. $x_{max} = y_{max} = 5$, $d_T = \frac{1}{2}$ and $c_x = c_y = 2.5$.

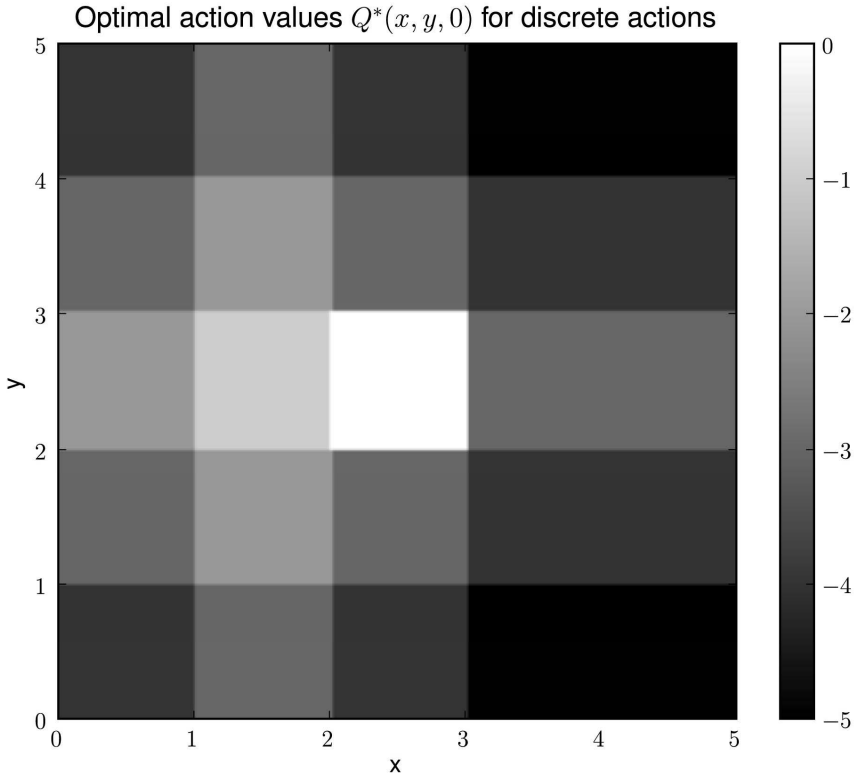


Figure 5. Action values for all states when choosing direction → (or φ=0) as first action and following an opti- mal policy with discrete actions afterwards. The corresponding state value function is depicted in Figure 3. Values rank from −5 to 0.

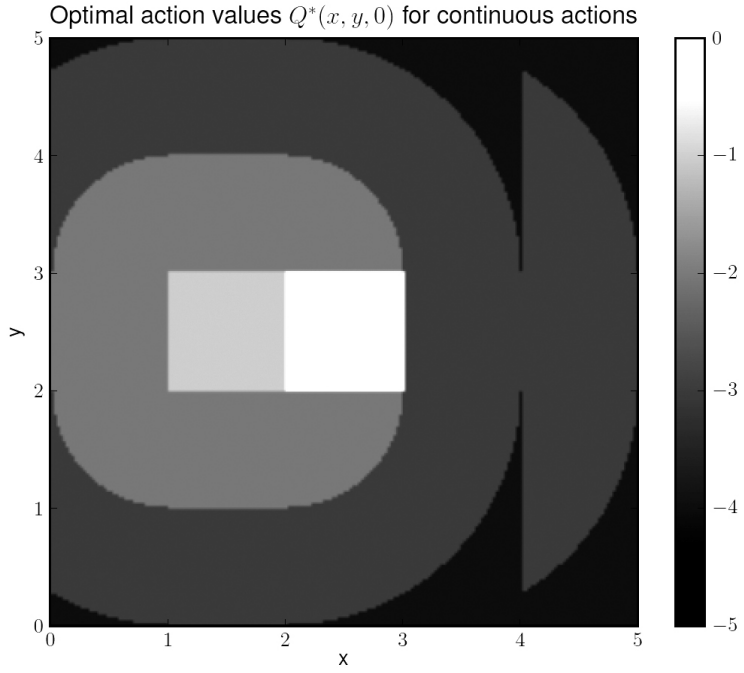


Figure 6. Action values for all states when choosing direction → (or φ=0) as first action and following an optimal policy for continuous actions afterwards. The according state value function is depicted in Figure 4. Values rank from −4 to 0.

3.4 Action Values

The problem as defined above is deterministic. With $V^*(x, y)$ given and $\gamma = 1$, the evaluation of the corresponding action value function is as simple as calculating the state value of the subsequent state (x', y') by means of (9) and (10), except for absorbing states:

$$Q^*(x,y,\varphi) = \begin{cases} 0 & \text{if(x,y)isterminal,} \\ V^*(x',y') - 1 & \text{otherwise,} \end{cases} \quad \textbf{(17)}$$

where the next state (x', y') is defined by (10). Its optimal state value $V^*(x', y')$ is computed through evaluation of (12) for discrete actions or (16) for continuous actions.

For the direction →, or φ=0, the action value function $Q^*(x, y, \rightarrow)$ is depicted in Figure 5 for four discrete angles (→, ↑, ←,↓) and in Figure 6 for continuous angles. Disregarding the target area both plots can be described by $V^*(s)-1$ shifted to the left with a respective copy of the values for $x \in]3,4]$ to $x \in]4,5]$.

4. Algorithm for Searching Directions with Reinforcement Learning

In the following an algorithm is presented which is capable of solving the direction finder problem for discrete and continuous actions. It is based on a linear approximation $\tilde{Q}$ of the action value function Q and is often used and well-known for discrete actions. Note, that we have slightly changed the notation in order to apply ideas from state generalisation to the generalisation of actions, which enables us to handle continuous actions.

4.1 Handling of Discrete and Continuous Actions

When searching for a control for tasks with a small finite number of actions (*discrete actions)*, it is possible to assign an individual approximation function $\tilde{Q}(s,a) \equiv \tilde{Q}_a(s)$ to every possible action. When using linear approximation, this is often implemented by using a fixed feature vector for every state s independent of an action a and a separate weight vector $\boldsymbol{\theta}_a$ for every action:

$$\tilde{Q}_a(s) = \boldsymbol{\theta}_a{}^{\mathrm{T}}\, \boldsymbol{\phi}(s) \quad \textbf{(18)}$$

For a very large set of possible actions, e.g. if the action is described by (dense) real-valued numbers drawn from an interval (*continuous actions*), this approach reaches its limits. In addition, the operation $\max_{a'} Q(s,a')$ becomes very expensive when implemented as brute force search over all possible actions and can hardly be performed within a reasonable time.

There are many ideas for dealing with continuous actions in the context of Reinforcement Learning. Examples are the *wirefitting* approach [14], the usage of an *incremental topology preserving map* building an average value from evaluating different discrete actions [15] and the application of neural networks (e.g. [16]).

To the best of our knowledge, so far no method has been shaped up as the algorithm of choice; so we want to contribute another idea which is rather simple and has been successfully applied to the direction finder problem. The basic idea, which is more a building block than a separate method, is not only to use feature vectors for state generalisation, but also to calculate separate feature vectors for actions. We combine state and action features in such a manner that each combination of feature components for a state and for an action uses a separate weight. If one has a combined feature $\boldsymbol{\phi}(s,a)$, the linear approximation can be expressed as

$$\tilde{Q}(s,a) = \boldsymbol{\theta}^{\mathrm{T}}\, \boldsymbol{\phi}(s,a) \quad \textbf{(19)}$$

where each weight, a component of $\boldsymbol{\theta}$, refers to a combination of certain sets of states and actions. This raises the question how to build a combined feature vector $\boldsymbol{\phi}(s,a)$. Here, two possibilities are considered.

Having a state feature $\boldsymbol{\phi}^{\mathcal{S}}(s) \in \mathbb{R}^{n_s}$ and an action feature $\boldsymbol{\phi}^{\mathcal{A}}(a) \in \mathbb{R}^{n_a}$, with components

$$\boldsymbol{\phi}^{\mathcal{S}}(s) = \left(\phi_1^{\mathcal{S}}(s), \ldots, \phi_{n_s}^{\mathcal{S}}(s)\right) \text{ and} \quad \textbf{(20)}$$

$$\boldsymbol{\phi}^{\mathcal{A}}(a) = \left(\phi_1^{\mathcal{A}}(a), \ldots, \phi_{n_a}^{\mathcal{A}}(a)\right), \quad \textbf{(21)}$$

an obvious way of combination would be to simply concatenate (*CC*) them:

$$\boldsymbol{\phi}^{\mathrm{CC}}(s,a) = (\boldsymbol{\phi}^{\mathcal{S}}(s), \boldsymbol{\phi}^{\mathcal{A}}(a)) \quad \textbf{(22)}$$

$$= \left(\phi_1^{\mathcal{S}}(s), \ldots, \phi_{n_s}^{\mathcal{S}}(s), \phi_1^{\mathcal{A}}(a), \ldots, \phi_{n_a}^{\mathcal{A}}(a)\right). \quad \textbf{(23)}$$

Concerning linear approximation, this would lead to a separation in weights for states and actions which is not suitable for estimating $Q(s, a)$ in general, because the coupling of states and actions is lost:

$$\tilde{Q}^{\mathrm{CC}}(s,a) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}^{\mathrm{CC}}(s,a) = \boldsymbol{\theta}_1{}^{\mathrm{T}}\boldsymbol{\phi}^{\mathcal{S}}(s) + \boldsymbol{\theta}_2{}^{\mathrm{T}}\boldsymbol{\phi}^{\mathcal{A}}(a). \quad (24)$$

Alternatively, one can form a matrix product (*MP*) of the feature vectors which results in a weight for every combination of components of the vectors $\boldsymbol{\phi}^{\mathcal{S}}(s)$ and $\boldsymbol{\phi}^{\mathcal{A}}(a)$ building a new feature vector:

$$\phi^{\mathrm{MP}}_{(i-1)n_a+j}(s,a) = (\boldsymbol{\phi}^{\mathcal{S}}(s)\left[\boldsymbol{\phi}^{\mathcal{A}}(a)\right]^{\mathrm{T}})_{i,j}, \quad (25)$$

with $i = 1,\ldots,n_s, j = 1,\ldots,n_a$.

The resulting matrix has been reshaped to a vector by concatenating the rows in order to stick with the terminology of a feature vector. This kind of feature assembling has been used for the results in the following sections. From now on, the tag $^{\mathrm{MP}}$ is left out for better readability.

With this kind of combined feature vector, the known case of discrete actions with separate weights for each action can be expressed by means of (19). This is done by choosing the action's feature vector as

$$\boldsymbol{\phi}^{\mathcal{A}}(a) = (\delta_{a,\bar{a}_1},\ldots,\delta_{a,\bar{a}_m})^{\mathrm{T}} \quad (26)$$

with $a \in \mathcal{A}(s) \subset \mathcal{A} = \{\bar{a}_1,\ldots,\bar{a}_m\}$ and $\delta_{a,\bar{a}_i} = 1$ for $a = \bar{a}_i$ and 0 otherwise. This is equivalent to the usage of binary features with only one feature present, the latter corresponds to the action a.

For continuous actions, all methods to calculate feature vectors for states can also be considered to be used for actions. One has to choose some appropriate way to generalise every action by a feature vector.

There are several ideas for procedures to compute a feature vector. One of the most popular is the usage of tile coding in a *Cerebellar Model Articulation Controller* (CMAC) architecture, e.g. successfully applied for continuous states in [2]. A comparison of radial-based features (RBF) with CMAC for function approximation is given in [17]. In the following, we present different kinds of state and action features used for the direction finder.

4.2 State Features

One of the simplest feature types are *binary* features. For states of the direction finder, we used the square-shaped areas

$$\mathcal{I}_{i,j} = \left[\frac{i-1}{4},\frac{i}{4}\right] \times \left[\frac{j-1}{4},\frac{j}{4}\right] \quad (27)$$

for $i, j = 1,\ldots,20$ with centres

$$c^{\mathcal{S}}_{ij} = \left(\frac{2i-1}{8},\frac{2j-1}{8}\right). \quad (28)$$

For binary state features we evaluate

$$\phi^{\mathcal{S}}_{ij}(s) = \begin{cases} 1 & \text{if } s \in \mathcal{I}_{i,j}, \\ 0 & \text{otherwise.} \end{cases} \quad (29)$$

Here s denotes the tuple (x, y). Row by row, the components of (29) are combined to a feature vector $\boldsymbol{\phi}^{\mathcal{S}}(s) = (\phi^{\mathcal{S}}_1(s),\ldots,\phi^{\mathcal{S}}_{400}(s))^{\mathrm{T}}$.

Similarly, for *radial-based* state features we used

$$\tilde{\phi}^{\mathcal{S}}_{ij}(s) = \exp\left(-16(s - c^{\mathcal{S}}_{ij})^2\right). \quad (30)$$

Again, the components are combined to a feature vector $\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)$ which is additionally normalised afterwards:

$$\boldsymbol{\phi}^{\mathcal{S}}(s) = \frac{\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)}{\sum_l \tilde{\phi}^{\mathcal{S}}_l(s)}. \quad (31)$$

4.3 Action Features

For the set of discrete actions Equation (26) results in a binary feature vector

$$\boldsymbol{\phi}^{\mathcal{A}}(\varphi) = (\delta_{\varphi,\rightarrow},\delta_{\varphi,\uparrow},\delta_{\varphi,\leftarrow},\delta_{\varphi,\downarrow})^{\mathrm{T}}. \quad (32)$$

For continuous actions a, i.e. for arbitrary angles $\varphi \in [0, 2\pi]$, we use the sixteen components

$$\tilde{\phi}^{\mathcal{A}}_l(\varphi) = \exp\left(-\frac{64}{\pi^2}(\varphi - c^{\mathcal{A}}_l)^2\right) \quad (33)$$

with $l = 1,\ldots,16$ and $c^{\mathcal{A}}_l = \frac{2l-1}{16}\pi$. These components decay quite fast and can be regarded as zero when used at a distance of 2π from the centers $c^{\mathcal{A}}_l$ of the Gaussians:

$$\tilde{\phi}^{\mathcal{A}}_l(\varphi \pm 2n\pi) \approx 0 \;\forall l,\; \varphi \in [0,2\pi],\; n = 1,2. \quad (34)$$

This is used to build feature vectors accounting for the periodicity

$$Q^*(x,y,\varphi) = Q^*(x,y,\varphi \pm 2\pi),$$

which should be met by the approximation $\tilde{Q}(x,y,\varphi)$. In order to approximately achieve this, we combine the Gaussians as

$$\hat{\phi}_l^{\mathcal{A}}(\varphi) = \tilde{\phi}_l^{\mathcal{A}}(\varphi - 2\pi) + \tilde{\phi}_l^{\mathcal{A}}(\varphi) + \tilde{\phi}_l^{\mathcal{A}}(\varphi + 2\pi) \tag{35}$$

and define a normalised action feature vector

$$\boldsymbol{\phi}^{\mathcal{A}}(\varphi) = \frac{\hat{\boldsymbol{\phi}}^{\mathcal{A}}(\varphi)}{\sum_l \hat{\phi}_l^{\mathcal{A}}(\varphi)}, \tag{36}$$

which is finally used in (25) as $\boldsymbol{\phi}^{\mathcal{A}}(a)$.

Using Equations (34), (35), (25) and (19) it follows that

$$\begin{aligned}
&\hat{\phi}_l^{\mathcal{A}}(\varphi) = \hat{\phi}_l^{\mathcal{A}}(\varphi \pm 2\pi) \\
\Rightarrow\ &\boldsymbol{\phi}^{\mathcal{A}}(\varphi) = \boldsymbol{\phi}^{\mathcal{A}}(\varphi \pm 2\pi) \\
\Rightarrow\ &\boldsymbol{\phi}^{\mathcal{S}}(x,y)(\boldsymbol{\phi}^{\mathcal{A}}(\varphi))^{\mathrm{T}} = \boldsymbol{\phi}^{\mathcal{S}}(x,y)(\boldsymbol{\phi}^{\mathcal{A}}(\varphi \pm 2\pi))^{\mathrm{T}} \\
\Rightarrow\ &\boldsymbol{\phi}(x,y,\varphi) = \boldsymbol{\phi}(x,y,\varphi \pm 2\pi) \\
\Rightarrow\ &\tilde{Q}(x,y,\varphi) = \tilde{Q}(x,y,\varphi \pm 2\pi).
\end{aligned}$$

Feature vectors for continuous angles built like this are henceforth called *cyclic radial-based*. In order to have a second variant of continuous action features at hand, we directly combine the components $\tilde{\phi}_l^{\mathcal{A}}(\varphi)$ defined in (33) to a feature vector and normalise it in a way equivalent to (36). These features are denoted with the term *radial-based* and are not cyclic.

4.4 Sarsa(λ) Learning

For the solution of the direction finder control task, the *gradient-descent Sarsa*(λ) method has been used in a similar way as described in [13] for a discrete set of actions when accumulating traces. The approximation $\tilde{Q}$ of Q is calculated on the basis of (19) using (23) for $\boldsymbol{\phi}(s,a)$ with features as described in Subsections 4.2 and 4.3. The simulation started with $\boldsymbol{\theta}_{t=0} \equiv 0$. There were no random actions and the step-size parameter α for the learning process has been held constant at $\alpha = 0.5$.

Since the gradient of $\tilde{Q}$ is $\boldsymbol{\phi}(s,a)$, the update rule can be summarised as

$$\begin{aligned}
\delta_t &= r_{t+1} + \gamma \tilde{Q}(s_{t+1}, a_{t+1}; \boldsymbol{\theta}_t) - \tilde{Q}(s_t, a_t; \boldsymbol{\theta}_t)\, \boldsymbol{\phi}(s) \\
&= r_{t+1} + \boldsymbol{\theta}_t^{\mathrm{T}}(\gamma \boldsymbol{\phi}(s_{t+1}, a_{t+1}) - \theta(s_t, a_t))
\end{aligned} \tag{37}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \delta_t \boldsymbol{e}_t \tag{38}$$

$$\boldsymbol{e}_{t+1} = \gamma \lambda \boldsymbol{e}_t + \boldsymbol{\phi}(s_{t+1}, a_{t+1}) \tag{39}$$

The vector $\boldsymbol{e}_t$ holds the eligibility traces and is initialised with zeros for $t = 0$ like $\boldsymbol{\theta}_t$. An episode stops when the agent encounters a terminal state or when 20 steps are taken.

4.5 Action Selection

Having an approximation $\tilde{Q}(s,a)$ at hand, one has to provide a mechanism for selecting actions. For our experiments with the direction finder, we simply use the greedy selection

$$\pi(s) = \arg\max_a \tilde{Q}(s,a). \tag{40}$$

For discrete actions, $\pi(s)$ can be found by comparing the action values for all possible actions. For continuous actions in general, the following issues have to be considered:

1) standard numerical methods for finding extrema normally search for a local extremum only,

2) the global maximum could be located at the boundary of $\mathcal{A}(s)$, and

3) the search routine operates on an approximation $\tilde{Q}(s,a)$ of $Q(s,a)$.

For the results of this paper, the following approach has been taken: The first and the second issue are faced by evaluating $\tilde{Q}(x,y,\varphi)$ at 100 uniformly distributed actions besides φ=0 and φ=2π, taking the best candidate in order to start a search for a local maximum. The latter has been done with a Truncated-Newton method using a conjugate-gradient (TNC), see [18]. In general, in the case of multidimensional actions a, the problem of global maxima at the boundary of $\mathcal{A}(s)$ may be tackled by recursively applying the optimisation procedure on subsets of $\mathcal{A}(s)$ and fixing the appropriate components of a at the boundaries of the subsets. The third issue, the approximation, cannot be compassed, but allows for some softness in the claim for accuracy, which may be given as an argument to the numerical search procedure.

5. Simulation Results

In the following, the direction finder is solved by applying variants of the Sarsa(λ) algorithm, which differ in the choice of features for states and actions. Each of them is regarded as a separate algorithm whose effectiveness concerning the direction finder is expressed by a single number, the so-called *detour coefficient*. In order to achieve this, the total rewards obtained by the agent are related to the optimal state values $V^*(s_0)$, where s_0 is the randomly chosen starting state varying from episode to episode.

5.1 Detour Coefficient

Let ε be a set of episodes. We define a measure called detour coefficient $\eta(\varepsilon)$ by

$$\eta(\mathcal{E}) := \frac{1}{|\mathcal{E}|}\sum_{e\in\mathcal{E}}\left(V^*(s_0^e) - R^e\right). \qquad (41)$$

Here, R^e is the total (discounted) reward obtained by the agent in episode e and s_0^e is the respective initial state. The quantity $|\varepsilon|$ denotes the number of episodes in ε. In our results the $|\varepsilon|$=1000 last *non-trivial* episodes have been taken into account to calculate the detour coefficient by means of (41), whereas non-trivial means that the episodes start with a non-terminal state, so the agent has to decide for at least one action. For V^* one has to choose from Equations (12) and (16) according to the kind of actions used.

The smaller $\eta(\varepsilon)$, the more effective a learning procedure is to be considered. For the direction finder, the value of $\eta(\varepsilon)$ can be interpreted as the average number of additional steps per episode the RL agent needs in comparison to an omniscient agent. It always holds that $\eta(\varepsilon)\geq 0$, because the optimal solution cannot be outperformed.

The detour coefficient depends on the random set of starting states of the episodes $e\in\varepsilon$, therefore we consider $\eta(\varepsilon)$ as an estimator and assign a standard error for the mean by evaluating

$$s_\eta(\mathcal{E}) = \sqrt{\frac{1}{|\mathcal{E}|\,(|\mathcal{E}|-1)}\sum_{e\in\mathcal{E}}(\eta_e - \eta(\mathcal{E}))^2}, \qquad (42)$$

with $\eta_e = V^*(s_0^e) - R^e$.

For the following results, the set of randomly chosen starting states is the same for the algorithms compared to each other.

5.2 Discrete Actions

For discrete actions, we have solved the direction finder problem for two different settings, with binary state features according to (29) and with radial-based state features according to (30) and (31). The other parameters were: 10000 episodes, $\alpha = 0.5$, $\lambda = 0.3$, $T_{\max} = 20$. Calculating the detour coefficient (41) for the last 1000 episodes not having a terminal initial state we observe $\eta(\varepsilon)$=0 for binary state features. This means, these episodes have an optimal outcome. Repeating the simulation with radial-based state features, there were six episodes of the last 1000 non-trivial episodes which in total took 12 steps more than needed, so $\eta(\varepsilon) = 0.012 \pm 0.005$. The respective approximation $\tilde{Q}(x,y,\varphi)$ after 10000 episodes for radial-based state features is shown in Figure 7a as image plot which is overlayed by a vector plot visualising the respective greedy angles. An illustration of the difference $V^*(x,y) - \max_\varphi \tilde{Q}(x,y,\varphi)$ for sample points on a grid is depicted in Figure 7b. Due to (6) such a plot can give a hint, where the approximation is inaccurate. Here the discontinuities of V^* stand out.

Both kinds of features are suitable in order to find an approximation $\tilde{Q}$ from which a good policy can be deduced, but the binary features perform perfectly in the given setting, which is not surprising because the state features are very well adapted to the problem.

5.3 Continuous Actions

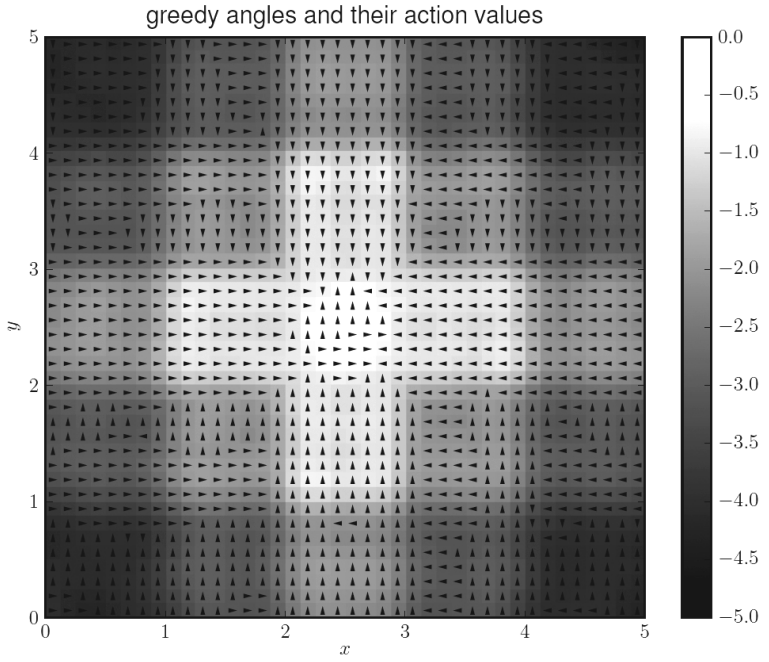
For continuous actions, which means $\varphi\in[0,2\pi[$, solutions of the direction finder problem have been obtained for four different combinations of state and action features. The feature vectors have been constructed as described in Subsections 4.2 and 4.3. The results are summarized in Table 1.

For all simulations the following configuration has been used: $\alpha = 0.5$, $\lambda = 0.3$, maximum episode time $T_{\max} = 20$, and 20000 episodes. The values of $\eta(\varepsilon)$ and $s_\eta(\varepsilon)$ have been calculated as discussed in Subsection 5.1. Like for binary state features and discrete actions, the last 1000 non-trivial episodes did not always result in an optimal total reward, so $\eta(\varepsilon)$>0 for every algorithm. Our conclusion with respect to these results for the given configuration and control task is:

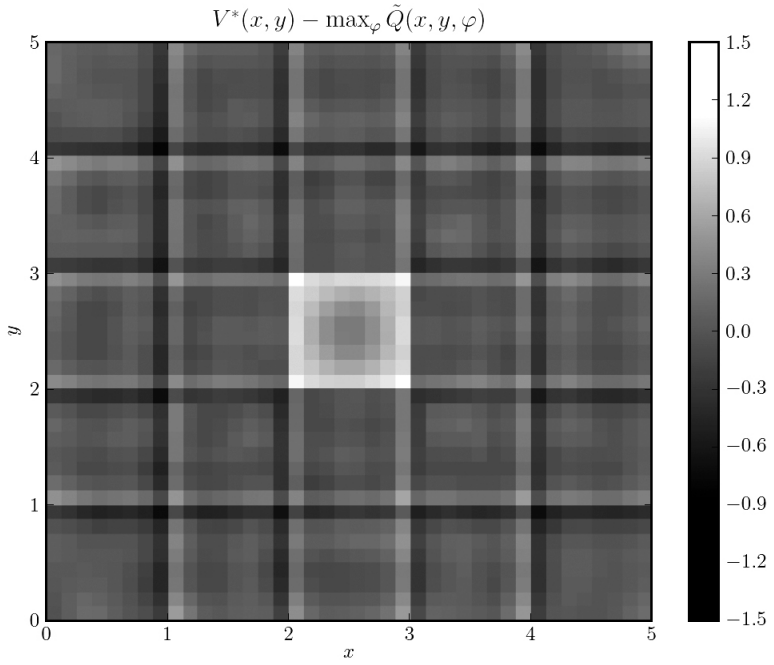
- Radial-based state features are more effective than binary state features.
- When using radial-based state features, cyclic radial-based action features outperform radial-based features lacking periodicity.

Table 1. Results for continuous actions for different combinations of state and action features: *bin* (binary), *rb* (radial-based), *crb* (cyclic radial-based).

state features	action features	$\eta(\varepsilon)$	$s_\eta(\varepsilon)$
bin	rb	0.109	0.011
bin	crb	0.115	0.013
rb	rb	0.090	0.011
rb	crb	0.063	0.009

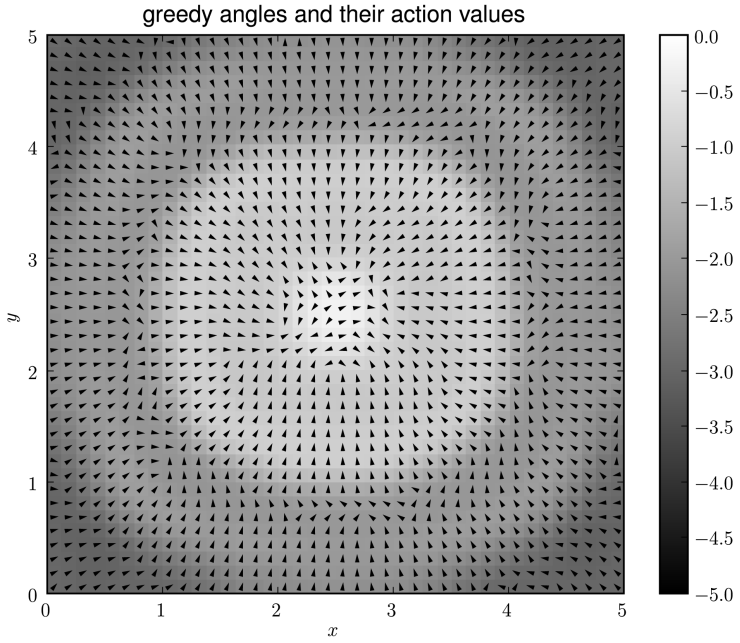


(a)

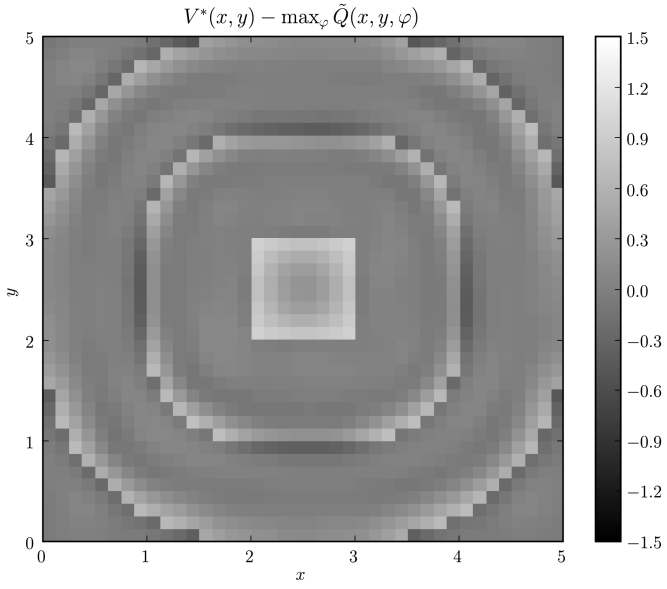


(b)

Figure 7. Results for learning with discrete actions, calculated by performing 10000 episodes, using radial-based state features. Possible directions: $\varphi \in \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$. For parameters, see Subsection 5.2. For these plots, the value functions and the policy have been evaluated on 40x40 points on a regular grid in the state space. (a) Discrete greedy actions $\varphi_g = \arg\max_\varphi \tilde{Q}(x,y,\varphi)$ and their values $\tilde{Q}(x,y,\varphi_g)$. The angles corresponding to the greedy actions are denoted by arrowheads. (b) Difference between $V^*(x,y)$ and $\max_\varphi \tilde{Q}(x,y,\varphi)$. The largest deviations are at the lines of discontinuity of $V^*(x,y)$. Compare to Figure 3.



(a)



(b)

Figure 8. Results for learning with continuous actions, calculated by performing 20000 episodes, using radial- based state features and cyclic radial-based action features. In principal, all directions are selectable: $\varphi \in [0, 2\pi[$. For parameters see Subsection 5.3. Like in Figure 7, for the sampling of $\max_\varphi Q(x,y,\varphi)$ a regular grid of 40x40 points has been used. (a) Continuous greedy actions $\varphi_g = \arg\max_\varphi \tilde{Q}(x,y,\varphi)$ and their values $\tilde{Q}(x,y,\varphi_g)$. The angles corresponding to the greedy actions are denoted by arrowheads. (b) Difference between $V^*(x,y)$ and $\max_\varphi \tilde{Q}(x,y,\varphi)$. The largest deviations are located near the discontinuities of $V^*(x,y)$. Compare with Figrue 4.

Figure 8 illustrates the function $\max_\varphi \tilde{Q}(x,y,\varphi)$ and its comparison with $V^*(x, y)$ for radial-based state features and cyclic radial-based action features after 20000 episodes. Monitoring this difference during the learning process helps to understand where difficulties for the approximation scheme or the action selection arise. Here, like for discrete actions, the discontinuities of $V^*(x,y)$ stand out. In order to get additional information, one can add an estimation procedure for V^* resulting in an approximation $\tilde{V}$, e.g. by using the TD(λ) algorithm [13,19], which runs independently in parallel to the learning process. Inspecting the evolution of $\tilde{V}$ supports an assessment about the covering of the state space.

6. Conclusions and Outlook

The direction finder is a control task in the domain of real-valued state spaces which allows choosing either discrete or continuous actions. For both cases, the optimal value functions V^* and Q^* are known analytically and can be compared with their approximations in detail. The optimal policy for continuous actions is not bang-bang-like and generally results in at least as good or better total rewards than the optimal policy used for four discrete actions. In general, the application of fine discretisations of the action space in order to find a continuous policy is not an option. In the case of the direction finder, one could think about using $8,16,32,\ldots$ discrete angles because the solution should converge to the solution for optimal angles, but the mere effort to find a greedy action increases linearly with the number of actions. In general, problems with continuous actions cannot be solved in a satisfying way by simply refining the methods for discrete actions but require new ideas. The development of these ideas is supported by evident control tasks like the direction finder.

Together with a measure like the detour coefficient (41), the direction finder can serve as a tool for testing new RL algorithms working on real-valued state spaces for both discrete and continuous actions allowing for intermediate comparison of the approximations $\tilde{V}$ and $\tilde{Q}$ with the known optimal value functions V^* and Q^*. The detour coefficient can also be used in combination with any other control task for which the optimal state value function V^* is known.

In order to solve the control task for continuous actions, we have introduced feature vectors for actions in order to approximate $\tilde{Q}(s,a)$. This may serve as a building block in other Reinforcement Learning algorithms.

The proposed algorithm has been applied to the direction finder control task for discrete and continuous actions using different combinations of features. In the case of continuous actions, we are able to infer from the detour coefficient that certain kinds of features are more suitable than others.

There are many improvements which can be applied to the search algorithm. So far, when searching for $\arg\max_a \tilde{Q}(s,a)$, the linearity of $\tilde{Q}$ together with the knowledge of the feature calculation scheme has not been taken into account. This may result in faster and more accurate optimisation. In addition, it would be interesting to use the detour coefficient to compare the algorithm with completely different approaches for continuous actions or other kinds of action selection.

Based on the known optimal solution, a debugging procedure can be defined, which helps to ensure that the basic capabilities of a new algorithm are present. Assuming the existence of an interface which allows switching between control tasks without changing the implementation of the algorithm, the implementation can be debugged with the presented direction finder control problem whose solution is exactly known in detail.

Such a debugging procedure can be composed of five subsequent steps:

1) Implement the proposed control task, the direction finder, along with functions to calculate $V^*(s)$ and $Q^*(s,a)$.
2) Prove the action selection capabilities by applying it to Q^* and compare the outcome of several runs with $V^*(s_0)$.
3) Prove the approximation abilities by "setting" the current approximation $\tilde{Q}$ of the action value function to Q^* as closely as possible, e.g. by using minimal least- squares.
4) Prove the stability of the algorithm by starting with the approximation of Q^* from the previous step, performing several learning cycles and testing for similarity of $\max_a Q^*(s,a)$ and $V^*(s)$.
5) Prove the convergence of the algorithm by starting with sparse or no previous knowledge and searching a "good" approximation $\tilde{Q}(s,a)$ for $Q^*(s,a)$.

These checks can be evaluated by comparing $\max_a Q^*(s,a)$ or $\max_a \tilde{Q}(s,a)$ with $V^*(s)$ or using a measure like the detour coefficient as defined in (39). If an

algorithm fails to pass this procedure, it will probably fail in more complex control tasks.

The direction finder is an example problem suitable for the testing of algorithms which are able to work on two-dimensional continuous state spaces, but there are more use cases, e.g. with more dimensions of state and/or action spaces. A collection of similar control problems with known optimal solutions (e.g. more complex or even simpler tasks) together with a debugging procedure can comprise a toolset for testing and comparing various Reinforcement Learning algorithms and their configurations respectively.

7. Acknowledgment

We would like to thank Prof. J. Honerkamp for fruitful discussions and helpful comments.

REFERENCES

[1] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in Advances in Neural Information Processing Systems 7, The MIT Press, pp. 369−376, 1995.

[2] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in Advances in Neural Information Processing Systems, edited by David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, The MIT Press, Vol. 8, pp. 1038−1044, 1996.

[3] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learning in continuous spaces," in ICML'00: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., pp. 903−910, 2000.

[4] M. G. Lagoudakis, R. Parr, and M. L. Littman, "Least-squares methods in reinforcement learning for control," in Proceedings of Methods and Applications of Artificial Intelligence: Second Hellenic Conference on AI, SETN 2002, Thessaloniki, Greece, Springer, pp. 752−752, April 11−12, 2002.

[5] K. Doya, "Reinforcement learning in continuous time and space," Neural Computation, Vol. 12, pp. 219−245, 2000.

[6] P. Wawrzyński and A. Pacut, "Model-free off-policy reinforcement learning in continuous environment," in Proceedings of the INNS-IEEE International Joint Conference on Neural Networks, pp. 1091−1096, 2004.

[7] J. Morimoto and K. Doya, "Robust reinforcement learning," Neural Computation, Vol. 17, pp. 335−359, 2005.

[8] G. Boone, "Efficient reinforcement learning: Model-based acrobot control," in International Conference on Robotics and Automation, pp. 229−234, 1997.

[9] X. Xu, D. W. Hu, and X. C. Lu, "Kernel-based least squares policy iteration for reinforcement learning," IEEE Transactions on Neural Networks, Vol. 18, pp. 973−992, 2007.

[10] J. C. Santamaría, R. S. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," Adaptive Behavior, Vol. 6, pp. 163−217, 1997.

[11] J. D. R. Millán and C. Torras, "A reinforcement connectionist approach to robot path finding in non-maze-like environments," Machine Learning, Vol. 8, pp. 363−395, 1992.

[12] T. Fukao, T. Sumitomo, N. Ineyama, and N. Adachi, "Q-learning based on regularization theory to treat the continuous states and actions," in the 1998 IEEE International Joint Conference on Neural Networks Proceedings, IEEE World Congress on Computational Intelligence, Vol. 2, pp. 1057−1062, 1998.

[13] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction of adaptive computation and machine learning," The MIT Press, March 1998.

[14] L. C. Baird and A. H. Klopf, "Reinforcement learning with high-dimensional, continuous actions," Technical Report, WL-TR-93-1147, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993.

[15] J. D. R. Millán, D. Posenato, and E. Dedieu, "Continuous-action q-learning," Machine Learning, Vol. 49, pp. 247−265, 2002.

[16] H. Arie, J. Namikawa, T. Ogata, J. Tani, and S. Sugano. "Reinforcement learning algorithm with CTRNN in continuous action space," in Proceedings of Neural Information Processing, Part 1, Vol. 4232, pp. 387−396, 2006.

[17] R. M. Kretchmar and C. W. Anderson, "Comparison of CMACS and radial basis functions for local function approximators in reinforcement learning," in International Conference on Neural Networks, pp. 834−837, 1997.

[18] R. Dembo and T. Steihaug, "Truncated-Newton algorithms for large-scale unconstrained optimization," Mathematical Programming, Vol. 26, pp.190−212, 1983.

[19] R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine Learning, Vol. 3, pp. 9−44, 1988.

