Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

On-line Reinforcement Learning for Nonlinear Motion Control: Quadratic and Non-Quadratic Reward Functions

Jan-Maarten Engel and Robert Babuška

Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628CD, Delft, The Netherlands

Abstract: Reinforcement learning (RL) is an active research area with applications in many fields. RL can be used to learn control strategies for nonlinear dynamic systems, without a mathematical model of the system being required. An essential element in RL is the reward function, which shows resemblance to the cost function in optimal control. Analogous to linear quadratic (LQ) control, a quadratic reward function has been applied in RL. However, there is no analysis or motivation in the literature, other than the parallel to LQ control. This paper shows that the use of a quadratic reward function in on-line RL may lead to counter-intuitive results in terms of a large steady-state error. Although the RL controller learns well, the final performance is not acceptable from a control-theoretic point of view. The reasons for this discrepancy are analyzed and the results are compared with non-quadratic functions (absolute value and square root) using a model learning actor-critic with local linear regression. One of the conclusions is that the absolute-value reward function reduces the steady-state error considerably, while the learning time is slightly longer than with the quadratic reward.

Keywords: reinforcement learning, reward function, linear quadratic control, control performance measure, steady-state error

1. INTRODUCTION

Reinforcement learning (RL) is an active research area in artificial intelligence, machine learning and optimal control. It has applications to real-world problems ranging from robotics to economics and medicine, as shown by (Peters et al., 2003; Godfrey and Powell, 2002; Ernst et al., 2006). In control, RL can be used to learn optimal control laws for nonlinear dynamic systems, without relying on a mathematical model of the system to be controlled. One of the primary elements in RL is the reward function. The reward function assigns a scalar reward to each state transition and the learning agent selects control actions so that the total cumulative reward over time is maximized. In this way, the reward function conveys information to the agent about the goal to be achieved.

The reward function plays a role similar to the cost function in control theory. In analogy to the linear quadratic regulator (LQR), a quadratic function of the state has been used as a reward function in RL (Abbeel et al., 2007; Grondman et al., 2012). While the use of the quadratic cost function in LQR is justified mainly by the fact that an analytic solution can be found for linear systems, no such reason exists in RL which relies on numerical solutions.

Classical RL literature (Sutton and Barto, 1998) recommends that the reward function should focus on the goal to be achieved, rather than on some specific behavior leading to the goal. Information concerning the desired behavior of the system which is available in advance can be implemented via an initial policy. Later, several authors (Mataric, 1994; Ng et al., 1999; Buşoniu et al., 2010) argued that learning can be accelerated by providing more information to the agent via the reward function. Methods of deriving reward functions that speed up reinforcement learning have been developed, see for example (Marthi, 2007; Singh and Barto, 2009). However, literature linking the reward function to the final control performance of the closed-loop system is lacking.

In this paper, insights in the closed-loop system behavior are provided for an RL agent trained on-line with a quadratic reward function. It is shown that the quadratic reward function provides incentives that are both desirable and undesirable. The influence of the discount factor is discussed as it is used in RL, but not in LQR. Finally, the performance of non-quadratic rewards is evaluated experimentally, including the absolute-value and square-root functions. The paper is organized as follows: Section 2 introduces the necessary preliminaries of the RL framework. In Section 3 the quadratic reward function is defined and its behavior is shown in an example, followed by a detailed analysis in Section 4. Non-quadratic rewards are proposed and evaluated in Section 5. Finally, Section 6 concludes the paper.

2. REINFORCEMENT LEARNING PRELIMINARIES

The RL problem can be described as a Markov decision process (MDP). For the ease of notation, the deterministic MDP is introduced, defined by the tuple (X, U, f, ρ) with $X \in \mathbb{R}^n$ the state space, $U \in \mathbb{R}^m$ the action space, $f : X \times U \to X$ the state transition function and $\rho : X \times U \times$

$X \to \mathbb{R}$ the reward function. The system to be controlled is described by the state transition function $f : X \times U \to X$, which returns the state x_{k+1} that the system reaches from state x_k after applying action u_k. After each transition, the controller receives a scalar reward $r_{k+1} \in \mathbb{R}$, which is given by the reward function $r_{k+1} = \rho(x_k, u_k, x_{k+1})$. The actions are chosen according to the policy $\pi : X \to U$. The goal in RL is then to find a policy, such that the expected discounted sum of future rewards is maximized. This sum, also called the return, is stored in a value function $V^\pi : X \to \mathbb{R}$ and defined as:

$$V^\pi(x) = E\Big\{\sum_{i=0}^{\infty} \gamma^i \rho(x_i, \pi(x_i), x_{i+1}) \Big| x_0 = x, \pi\Big\} \qquad (1)$$

where $\gamma \in [0, 1)$ is the discount factor.

The RL method used in this paper is the model learning actor-critic (MLAC) algorithm proposed by Grondman et al. (2012). Figure 1 provides a schematic overview of an RL agent based on an actor-critic method.

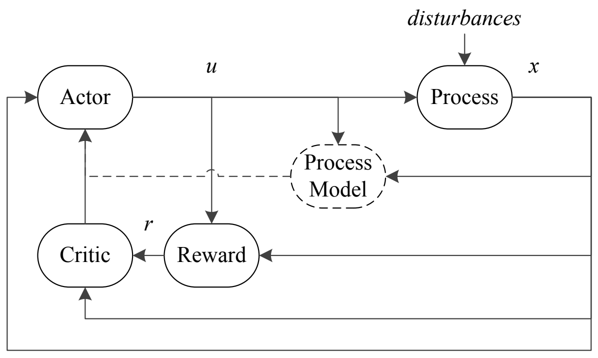

Fig. 1. Schematic representation of the model-learning actor-critic method.

The actor-critic technique is characterized by learning separate functions for the policy and the value function. The critic takes the role of the value function and evaluates the performance of the actor, as such assisting in the estimation of the gradient used for the actor's updates. In addition to the critic and the actor, MLAC uses an adaptive process model. The actor parameters are updated using the policy gradient calculated with the help of the process model. As both the state space and the action space are continuous, function approximators based on local linear regression (LLR) are applied. LLR is a non-parametric memory-based method for approximating nonlinear functions (Atkeson et al., 1997). Note that the learning proceeds in a series of finite trials, each of them consisting of N_s samples. After each trial, the system is reset to an initial state and the learning continues. Algorithm 1 gives the pseudo-code of the MLAC method.

3. QUADRATIC REWARD FUNCTION

The quadratic reward function in standard LQR is defined as follows:

$$J = \sum_{i=0}^{\infty} (x^d - x_i)^T Q (x^d - x_i) + u_i^T P u_i \qquad (2)$$

Algorithm 1 Pseudo-code for model-learning actor-critic

```
1: set parameters
2: initialize function approximators
3: **repeat**: for $N_t$ trials
4:     initialize state
5:     reset eligibility traces
6:     apply action
7:     $k \leftarrow 0$
8:     **repeat**: for $N_s$ time steps
9:         measure state
10:        % actor
11:        calculate action based on state
12:        apply action including exploration
13:        update actor
14:        % process model
15:        update process model
16:        % critic
17:        calculate reward and temporal difference
18:        update eligibility traces
19:        update critic
20:        $k \leftarrow k + 1$
21:    **until** $k = N_s$
22: **until** $j = N_t$
```

with x^d a constant desired state and Q, P positive definite weighting matrices of dimensions $n \times n$ and $m \times m$, respectively. These matrices are used to tune the relative contributions of the components of the state and action vectors to the cost. In analogy to (2), the reward function in RL is defined as:

$$\rho(x_{k+1}, u_k) = -(x^d - x_{k+1})^T Q (x^d - x_{k+1}) - u_k^T P u_k \quad (3)$$

Note that contrary to (2), the future state x_{x+1} is used in the reward. This is because the reward is given for the *transition to the next state*, rather than for being in the current state. The following expression for the return results:

$$\begin{aligned} \mathcal{R} &= \sum_{i=0}^{\infty} \gamma^i \rho(x_{i+1}, u_i) \\ &= -\sum_{i=0}^{\infty} \gamma^i \left[(x^d - x_{i+1})^T Q (x^d - x_{i+1}) + u_i^T P u_i\right] \end{aligned} \qquad (4)$$

As the convention in RL is to maximize the return, a negative sign is used with the quadratic form.

The weight P, whose role is to penalize excessive control actions, has to be used with care. If the desired state x^d corresponds to an equilibrium $f(x^d, u^d) = x^d$ with a nonzero control action u^d, (2) and also (4) will induce a trade-off between the steady-state error and the control action applied in x^d. As the focus in the analysis is on the (zero) steady-state error, $P = 0$ is used in the sequel.

There are two remarkable differences between the LQR cost function (2) and the quadratic return (4):

(1) The discount factor $\gamma \in [0, 1]$ is used in RL. While its primary purpose is to keep the sum in (4) finite, it also plays a role of a 'control horizon' and as such it has a considerable influence on the learning speed. Each application requires a different setting, which is a compromise between a long enough horizon to cover the expected response settling time (γ close to one)

and an acceptable learning speed which is favored by smaller values of γ.

(2) With finite learning trials, as in Algorithm 1, the infinite return (4) is effectively approximated by a finite sum:

$$\mathcal{R}_{N_s} = \sum_{i=0}^{N_s} \gamma^i \rho(x_{i+1}, u_i) \tag{5}$$

While this difference may seem unimportant (as long as the trial length N_s is sufficient to cover the expected response settling time), it will be shown in the remainder that it has a great influence on the final performance of the control policy learned.

For illustrative purposes, a specific motion-control example will be used throughout the remainder of the paper. A simple yet representative nonlinear control problem is chosen, which allows for easy reproduction of the results by an interested reader.

Example: Consider a 1-DOF robot arm, shown in Fig. 2, controlled to a specific angular position, under the influence of an unknown force due to gravity acting on the payload M.

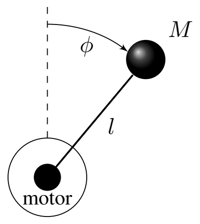

Fig. 2. 1-DOF robot arm with an unknown payload.

The dynamics of the system (not available to the RL agent) are given by

$$J\dot{\omega} = Mgl\sin(\phi) - (b + \tfrac{K^2}{R})\omega + \tfrac{K}{R}u \tag{6}$$

and the associated model parameters are presented in Table 1.

Table 1. Robot arm and learning parameters

Model parameter	Symbol	Value	Unit
Arm and payload inertia	J	$1.91 \cdot 10^{-4}$	kgm^2
Payload mass	M	$5.50 \cdot 10^{-2}$	kg
Acceleration due to gravity	g	9.81	m/s^2
Robot arm length	l	$4.20 \cdot 10^{-2}$	m
Damping	b	$3 \cdot 10^{-6}$	Nms
Torque constant	K	$5.36 \cdot 10^{-2}$	Nm/A
Rotor resistance	R	9.50	Ω
Sampling period	T_s	0.02	s
Trial duration	T_t	2	s
Samples per trial	N_s	100	$-$
Trials per experiment	N_t	200	$-$
Number of experiments	N_e	50	$-$

The state of the underlying Markov decision process is fully described by the angular position ϕ and the angular velocity ω

$$x = \begin{bmatrix} \phi \\ \omega \end{bmatrix} \tag{7}$$

and the action u is the motor voltage ranging from -10 V to $+10$ V. The goal of the RL agent is to learn to bring the payload from the pointing down position ($x_0 = [\pi, 0]^T$) to a desired position ($x^d = [\pi/2, 0]^T$) with a fast transient and no steady-state error. The input weighting matrix in the reward function (3) is therefore set to $P = 0$ and the associated Q-matrix is

$$Q = \begin{bmatrix} q_\phi & 0 \\ 0 & q_\omega \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}$$

The weights $q_\phi = 5$ and $q_\omega = 0.1$ were chosen to scale the angle and angular velocity to approximately the same range. The MLAC algorithm, Algorithm 1, is applied online and a step response of the system after learning is plotted in Fig. 3. Observe that the RL agent learned to bring the arm quickly to the vicinity of the desired position, but that a significant steady-state error remains.

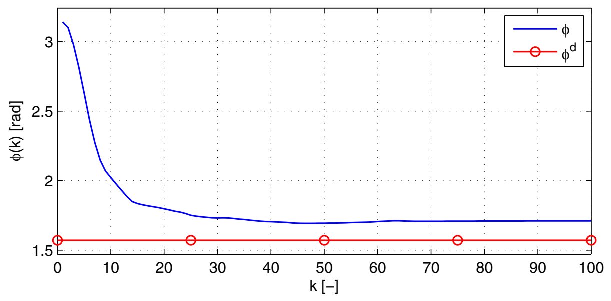

Fig. 3. A typical step response after learning with the quadratic reward function.

As the input weight $P = 0$, the steady-state error comes as a surprise. In the following section it is shown in detail what causes this behavior.

4. ANALYSIS OF THE QUADRATIC REWARD FUNCTION

The first important fact to keep in mind is that the reward function not only induces the final performance after learning, but it also has a significant influence on the convergence of the learning process itself. In terms of the final performance in a typical motion-control problem, the reward function yields a trade-off between the quick decrease of the position error and the velocity of the motion. A relatively large weight on the velocity will make the agent reduce the velocity, eventually leading to a slower response. However, setting the velocity weight q_ω to very low values, or even to zero, is not possible as this slows down the learning. So, in selecting the weights, one has to make a tradeoff between the desired final performance and the speed of the learning. Interestingly, the existence of a strong interaction between the target behavior and the success of the reinforcement learning process is well known in the area of human and animal training (Pryor, 2009).

Secondly, as the response is dynamic, the influence of the individual components of the reward function on the functional (4) is time dependent and nontrivial to predict. The discounting has a significant influence, too. In order to see these effects, the return is split into its position and velocity components. To take into account its finite approximation due to the finite trial duration, the cumulative reward computed up to sample k is expressed as:

$$\begin{aligned} R^{\gamma}(k) &= \sum_{i=0}^{k} -\gamma^{i}(x^{d}-x_{i+1})^{T}Q(x^{d}-x_{i+1}) \\ &= \sum_{i=0}^{k} -\gamma^{i}q_{\phi}(\phi^{d}-\phi_{i+1})^{2}-\gamma^{i}q_{\omega}(\omega^{d}-\omega_{i+1})^{2} \\ &= R^{\gamma}_{\phi}(k)+R^{\gamma}_{\omega}(k) \end{aligned} \tag{8}$$

with $R^{\gamma}_{\phi}(\cdot)$ and $R^{\gamma}_{\omega}(\cdot)$ the cumulative rewards for the individual components of the state vector. The superscript γ is added to emphasize the discounted nature of this performance measure. For the purpose of comparison, the undiscounted variant is defined as well: [1]

$$\begin{aligned} R(k) &= \sum_{i=0}^{k} -(x^{d}-x_{i+1})^{T}Q(x^{d}-x_{i+1}) \\ &= \sum_{i=0}^{k} -q_{\phi}(\phi^{d}-\phi_{i+1})^{2}-q_{\omega}(\omega^{d}-\omega_{i+1})^{2} \\ &= R_{\phi}(k)+R_{\omega}(k) \end{aligned} \tag{9}$$

To illustrate the contribution of the position error and velocity components over time, $R_{\phi}(\cdot)$ and $R_{\omega}(\cdot)$ are plotted in Fig. 4 for two different closed-loop step responses produced with two different policies $\pi(\cdot)$, here denoted as Policy 1 and 2.

Note that Policy 1 incurs a large steady-state error, whereas Policy 2 results in a negligible steady-state error. Yet, throughout almost the entire trial, the undiscounted cumulative reward for Policy 1 is higher than that of Policy 2, see the bottom left plot of Fig. 4. The reason for this discrepancy is the relatively large difference in $R_{\omega}(\cdot)$, due to the earlier decrease of the velocity with Policy 1 – noticeable only after a close inspection, see the downward peaks in the bottom right plot of Fig. 4. The contribution of the steady-state error to the cumulative reward slowly grows with time (as the constant error is integrated in the cumulative reward). So if the trial is sufficiently long, the steady-state error will eventually prevail. In the example, this will be beyond $k = 100$, see the bottom left plot of Fig. 4. This is, however, not true for the discounted cumulative reward shown in Fig. 5. The exponential discounting prevails over the approximately linear growth of the position error component and the cumulative reward becomes asymptotically constant. Therefore, the initial contribution due to the larger velocity obtained with Policy 1 can never be compensated by the error component of the position, even if the trial length would be extended. Recall that the RL agent uses the discounted cumulative reward as the learning criterion and it therefore may favor slower responses with a steady-state error to faster responses without the error, which is highly undesirable. Note that this is not the case with the LQR reward (2), which is undiscounted and in addition integrates the error over infinite time.

[1] Although the undiscounted cumulative reward does not correspond to the value function (1) which governs the learning, the undiscounted total reward per trial $R(N_s)$ is commonly used in the literature as a performance indicator for the speed of the learning process.

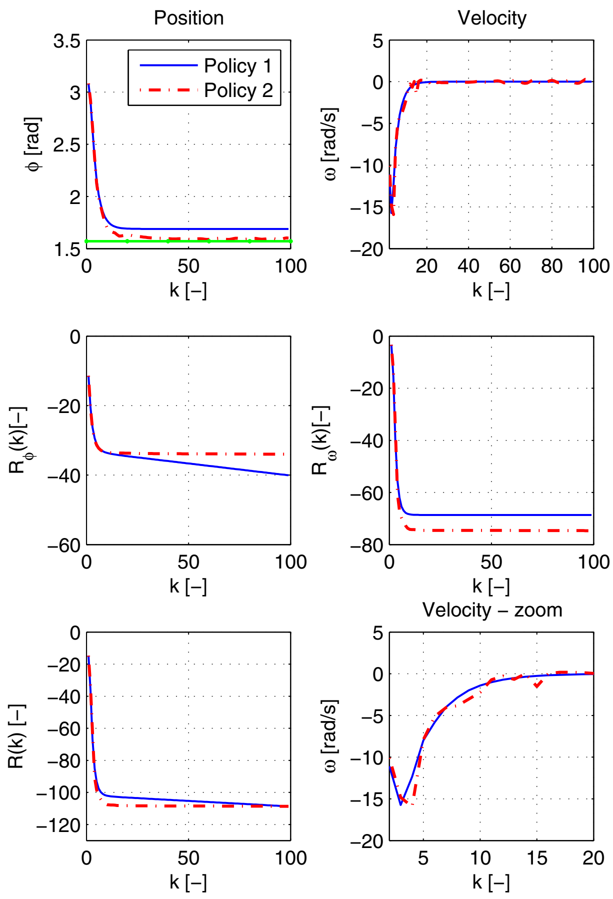

Fig. 4. Closed-loop step responses for two different policies and the corresponding undiscounted cumulative rewards.

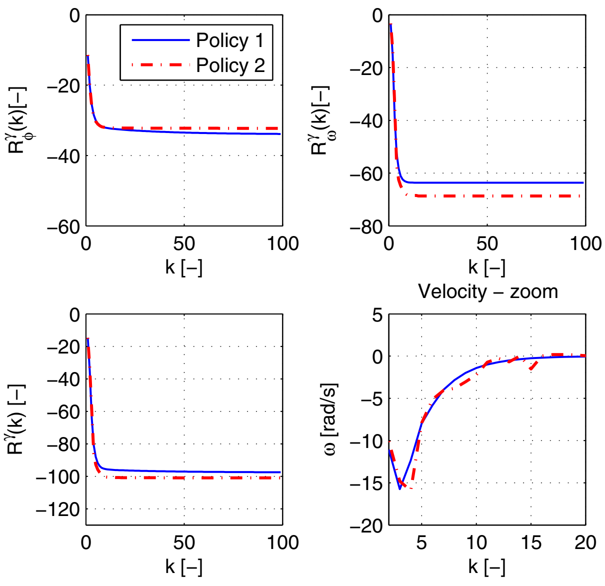

Fig. 5. The discounted cumulative rewards for the step responses shown in the top panel of Fig. 4.

One way to alleviate the above problem is to adjust the diagonal components of the Q matrix. This approach, however, has two main drawbacks: (i) it is impossible to set the weights correctly in advance, (ii) too low a weight on the velocity has a negative influence on the learning convergence. In addition, the weights scale the individual cumulative reward components linearly, while the components themselves grow quadratically with the state variables. As a result, the quadratic reward penalizes large velocities much more than small steady-state errors. A better solution is to modify the function itself, rather than the weights, as discussed in the following section.

5. NON-QUADRATIC REWARD FUNCTIONS

Instead of the quadratic function, rewards based on the absolute-value (10) and square-root (11) functions are suggested and evaluated in this section.

$$\rho_{||}(x) = -\left(q_\phi|\phi^d - \phi| + q_\omega|\omega^d - \omega|\right) \qquad (10)$$

$$\rho_{\surd}(x) = -\left(q_\phi\sqrt{|\phi^d - \phi|} + q_\omega\sqrt{|\omega^d - \omega|}\right) \qquad (11)$$

They both reduce the relative importance of the large transient error and increase the importance of the small steady-state error. Figure 6 shows these cumulative rewards computed for the step response of Fig. 3 and normalized to the range $[0, 1]$. These plots show that for the quadratic reward, the large initial error (approximately during the first 10 time steps, see Fig. 3), contributes to more than 90% of the final cumulative reward. The absolute-value and square-root reward reduce this contribution to about 60% and 40%, respectively. As a result, the contribution of the steady-state error to the overall cumulative reward increases.

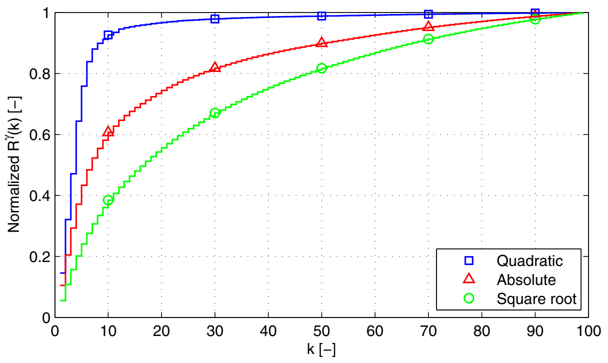

Fig. 6. Normalized discounted cumulative reward for the three different reward functions considered.

To see whether adjusting the reward function is beneficial in terms of both the learning speed and the final closed-loop behavior, the undiscounted return and the time response after learning are compared for the various reward functions.

Figure 7 shows the learning curve; the evolution of the undiscounted return per trial over the 200 trials that constitute the learning experiment. In order to compare the three norms, the return is normalized to the range $[0, 1]$. Furthermore, for the sake of quantitative comparison, the settling time and rise time of the learning curve are calculated for each reward function, see Appendix A.

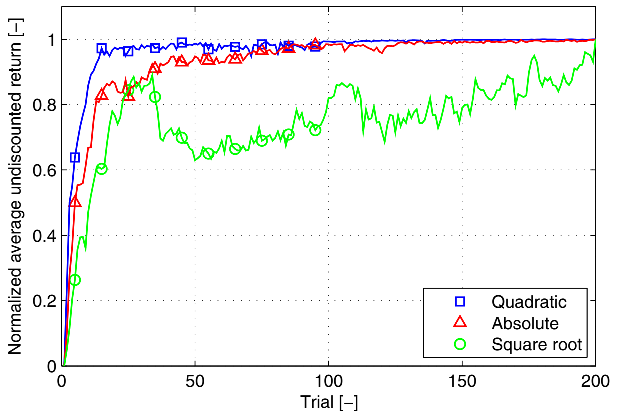

Fig. 7. Normalized average learning curve for the three rewards considered ($N_e = 50$).

The results in Table 2 show that the learning curve settling time for the quadratic reward is similar to the settling time of the absolute-value reward, whereas the rise time of the quadratic reward is significantly shorter compared to the absolute-value reward. Observe that the quadratic reward leads to slightly faster and more stable learning in comparison with the absolute-value reward, while the square-root reward results in unstable learning. Due to the unstable learning of the square-root reward no settling time and rise time are presented. The reasons for the faster learning with the quadratic reward and the unstable learning with the square-root reward will be investigated in our future research.

Table 2. Learning curve characteristics for the three rewards considered ($N_e = 50$).

Reward function	**Settling time** [seconds]	**Rise time** [seconds]
Quadratic	136	20
Absolute	142	60
Square root	n.a.	n.a.

Figure 8 shows the typical closed-loop step response after learning obtained with the three rewards. The absolute-value reward yields a response with a negligible steady-state error which is also faster than the one obtained with the quadratic reward. The response for the square-root reward also shows a fast transient, but the overall performance is worse. Some experiments with the square-root reward resulted in a negligible steady-state error, whereas other experiments showed persistent oscillations or steady-state errors. This is the subject of future research.

Table 3 shows the mean steady-state error per reward function, based on the 25 last samples of each experiment. The results endorse the findings previously discussed.

Table 3. Mean steady-state error for the three reward functions ($N_e = 50$).

Reward function	**Mean steady-state error**
Quadratic	0.1416
Absolute	0.0452
Square root	0.2492

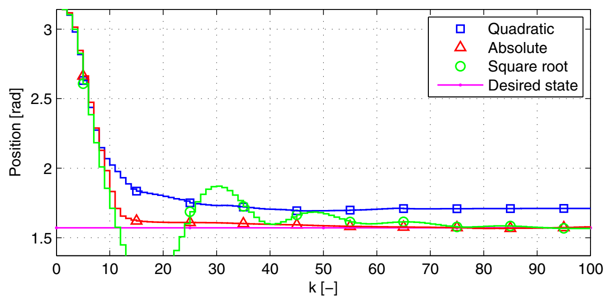

Fig. 8. Typical closed-loop step response after learning with the three reward functions considered.

In summary, when considering both the speed of learning and the final closed-loop response after learning, the absolute-value reward results in the best overall performance.

6. CONCLUSIONS AND FUTURE RESEARCH

In this paper, the performance of the quadratic reward function is analyzed in on-line reinforcement learning for nonlinear motion control. It is shown that this reward function provides insufficient incentives for the RL algorithm to find policies that achieve both fast response and zero steady-state error. This is due to the discounted nature of the return (the cost function minimized by the RL algorithm), as well as the amplification of large errors by the quadratic function. A reward function based on the absolute value of the state reduces the steady-state error, while the learning curve characteristics are similar to the quadratic reward. An exception to this is the learning curve rise time, which is significantly smaller for the quadratic reward. Furthermore, a reward function based on the square root of the state resulted in unstable learning. The reason for this behavior is unknown. Both the larger rise time for the absolute-value reward and the unstable learning for the square-root reward are subjects of future research.

Appendix A. LEARNING CURVE SETTLING TIME AND RISE TIME

To define the settling time and the rise time of the learning curve, first introduce the undiscounted return of trial j averaged over the number N_e of learning experiments:

$$\bar{R}_j = \frac{1}{N_e}\sum_{l=1}^{N_e} \mathcal{R}_{j,l_{N_s}} \tag{A.1}$$

where $\mathcal{R}_{j,l_{N_s}}$ is the undiscounted counterpart of the return defined in (5), referring to the jth trial within the lth learning experiment. For each reward, the sequence $\bar{R}_1, \bar{R}_2, \ldots, \bar{R}_{N_t}$ is normalized so that the maximum value of this sequence is 1.

The performance $\bar{R}_f$ at the end of learning is defined as the average normalized undiscounted return over the last c trials:

$$\bar{R}_f = \frac{1}{c}\sum_{j=N_t-c+1}^{N_t} \bar{R}_j \tag{A.2}$$

The settling time τ_s of the learning curve is then defined as the time elapsed from the beginning of the learning experiment till the moment at which the learning curve enters and remains within a band ϵ of the final value $\bar{R}_f$:

$$\tau_s = T_t \cdot \arg\max_j (|\bar{R}_f - \bar{R}_j| \geq \epsilon \bar{R}_f) \tag{A.3}$$

In this paper c and ϵ are set to 25 and 0.05 respectively.

The rise time is defined as the time required to climb from the 10% performance level to the 90% performance level:

$$\tau_{rise} = \tau_{90} - \tau_{10} \tag{A.4}$$

with τ_p defined as:

$$\tau_p = T_t \cdot \arg\max_j \left(\frac{\bar{R}_j - \bar{R}_1}{\bar{R}_f - \bar{R}_1} \geq \frac{p}{100} \right) \tag{A.5}$$

for $p = 10\%$ and 90%.

REFERENCES

Abbeel, P., Coates, A., Quigley, M., and Ng, A. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19*, 2007. MIT Press.

Atkeson, C.G., Moore, A.W., and Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, 11(1-5), 11–73.

Buşoniu, L., Babuška, R., De Schutter, B., and Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press.

Ernst, D., Stan, G.B., Gonalves, J., and Wehenkel, L. (2006). Clinical data based optimal STI strategies for HIV: a reinforcement learning approach. In *Decision and Control, 2006 45th IEEE Conference on*, 667–672.

Godfrey, G.A. and Powell, W.B. (2002). An adaptive dynamic programming algorithm for dynamic fleet management, I: Single period travel times. *Transportation Science*, 36(1), 21–39.

Grondman, I., Vaandrager, M., Buşoniu, L., Babuška, R., and Schuitema, E. (2012). Efficient model learning methods for actor-critic control. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics.*

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, 601–608. ACM, New York, NY, USA.

Mataric, M.J. (1994). Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, 181–189. Morgan Kaufmann.

Ng, A.Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287. Morgan Kaufmann.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robotics*, 1–20.

Pryor, K. (2009). *Don't Shoot the Dog! The New Art of Teaching and Training.* Ringpress Books, Surrey, UK, revised edition.

Singh, S. Lewis, R. and Barto, A. (2009). Where do rewards come from? *Annual Conference of the Cognitive Science Society*, 2601–2606.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction.* The MIT Press.