Continuous State Space Q-Learning for control of Nonlinear Systems

ten Hagen, S.H.G.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 1

Introduction

In everyday life we control many systems. We do this when riding a bicycle, changing the temperature in a room or driving in a city to a certain destination. Although these activities are quite different they have in common that we try to influence some physical properties of a system on the basis of certain measurements. In other words, we try to control a system.

Figure 1.1 shows the configuration of a control task. The system represents the physical system we have. Suppose we want to change the temperature in a room, then the room is the system. The properties of the system that do not change form the parameters of the system. The height of the room is such a parameter. The properties of the system that are not constant form the state of the system. So in our example the temperature is the state of the system.

The state can change, depending on the current state value and the parameters of the system. Because we want to control the system, we also should be able to influence the state changes. The influence we can have on the state changes are called the control actions. So by applying each time the right control actions we try to change the state to

\[ \text{Disturbance} \rightarrow \text{System} \rightarrow \text{Parameters} \rightarrow \text{State} \]

\[ \text{Control Action} \rightarrow \text{Controller} \]

Figure 1.1. The control configuration.
some desired value. In our example this means we have a heater that can be switched on and off. The disturbance in figure 1.1 indicates a change in system or state change that can not be controlled. The control actions have to be chosen such that they reduce the effect of the disturbance. So if the temperature in our room drops because a window is opened, the heater should be switched on.

Instead of applying all actions “by hand”, it is also possible to design an algorithm that computes the appropriate control actions given the state. This algorithm is called the controller. So based on the measurement of the temperature, the controller has to decide whether to switch the heater on or off. Before designing a controller, one has to specify the desired behavior of the system and controller. Then based on the desired behavior and all available information of the system, a controller is designed according to a certain design procedure. This design procedure provides a recipe to construct a controller. After the design one checks whether the design specifications are met.

A different approach to obtain a controller is to design a control architecture in which the controller itself learns how to control the system. In order to do this the architecture should incorporate a mechanism to interact with the system and store the gained experiences. Based on these experiences the controller should infer what for the control task the most appropriate action is. This seems a very natural approach, resembling the way we learn how to control.

Machine Learning (ML) is a subfield within Artificial Intelligence research, in which inference mechanisms are studied. From a ML point of view the inference mechanism in the above described control architecture is called Reinforcement Learning (RL). The control architecture receives a scalar evaluation, called the reinforcement, for each action applied to the system. In this way the consequence of actions can be associated with the states in which these actions were applied. In this thesis we will focus on Q-Learning, a particular kind of RL, in which the consequences of actions are associated for each state and action combination. Then for each state that action, for which the consequences are most desirable, can be selected.

In this chapter we will first formalize our control task and give an overview of different controller design approaches. We present some problems and show how they are solved. Then we will give a short description of RL. Finally we will present our problem statement and give an overview of the remainder of this thesis.

1.1 Control

If the control of a system has to be automated, a controller has to be designed. This normally is the task of control engineers. Control theory provides the mathematical framework in which the controller design task can be formulated. In this framework the real physical system is described by abstract representation called the model.

In order to find the controller, the model should indicate the influence of the control action on the state change. The function describing the state change is called the state transition. In this thesis we will only consider time discrete systems. If the state is
represented as vector \( x \in \mathbb{R}^{n_x} \), the control action as vector \( u \in \mathbb{R}^{n_u} \) and the noise as vector \( v \in \mathbb{R}^{n_v} \), the state transition is given by:

\[
x_{k+1} = f(x_k, u_k, v_k).
\] (1.1)

This is the model of the system. The \( k \) in (1.1) represents the time step. The \( f \) forms a functional mapping from \( \mathbb{R}^{n_x+n_u+n_v} \rightarrow \mathbb{R}^{n_x} \) that represents the state transition. The parameters of the system are included in \( f \).

In (1.1) the vector \( v \) can represent external disturbances that influence the state transition. But the model forms an abstract approximation of the system, that does not necessarily have to be complete. So \( v \) can also be used to take into account the consequences of the unmodeled dynamics of the system. In the design of the controller \( v \) is regarded as a stochastic variable.

Note that the model in (1.1) represents a Markov process. This means that the next state \( x_{k+1} \) does not depend on past states and actions. Throughout this thesis we will assume a model like (1.1). This is a reasonable assumption because all higher order Markov processes can be represented as zero-order Markov processes as in (1.1). This is possible by combining the past states that influence the state transition and use this as the new state. Then the model in (1.1) can be used to describe the change of the new state.

Another issue not reflected in the model of (1.1) are the measurements. In a real system the value of the state is not always available through measurements by sensors. Still measurements are made because they are needed to control the system. These measurements form an extra output of the system. To take this into account in the model in (1.1), an extra function has to be specified that maps \( x, u \) and \( v \) to the measurements. We will not include it in the model and so we will assume that the state values are measured directly.

The controller has not been described yet. Since we will only consider models like (1.1), we have to assume that the state value is available to the controller. Also we will assume that the controller only uses the present state value. This leaves as only possible candidate: a state feedback controller. This means that the controller is formed by a functional mapping from state to control action, so \( u = g(x) \). With these assumptions, designing the controller becomes a matter of finding the appropriate \( g \). The function \( g \) will be called the feedback.

### 1.1.1 Designing the state feedback controller

The goal of the design is to find the best \( g \) to control the system. This means that the desired behavior of the system plus controller should be specified. There are two main design principles:

- **Optimality**
  
  A design method based on optimal control optimizes some criterion. This criterion

\footnote{For continuous time systems the model represents the change of the state, rather than the new state value.}
gives a scalar evaluation of all state values and control actions. Traditionally, the evaluation represents costs, so the task is to find the feedback that leads to the lowest costs.

- **Stability**

A design method based on stability prevents the system from becoming instable. The state value of an instable system can grow beyond any bounds and potentially damage the real system.

The main difference between these design principles is that optimality based design aims at the best possible behavior, while a stability based design aims at the prevention of the worst possible behavior. Note that these principles do not exclude each other. For an instable system the costs would be very high, so an optimality based design would also try to stabilize the system. And if the controller stabilizes the system, then it is still possible to choose the controller that performs best given an optimal control criterion.

The two design principles still do not indicate how the feedback is computed. To explain this it is convenient to first introduce an important class of systems, the linear systems. The model of a linear system is given by (1.1) where \( f \) represents a linear function of the state, the control action and the noise:

\[
x_{k+1} = Ax_k + Bu_k + v_k.
\]  

(1.2)

The matrices \( A \) and \( B \) have the proper dimensions and represent the parameters of the linear system.

When a stability based design method is applied, a quantification of the stability can be used to select the parameters of the feedback. Suppose that the feedback is also linear:

\[ u = Lx. \]

(1.3)

where \( L \in \mathbb{R}^{n_u \times n_x} \) represents the linear feedback. This makes it possible to describe the state transition based on the closed loop. Applying (1.3) to (1.2) gives\(^2\) \( x_{k+1} = (A + BL)x_k = Dx_k \), where \( D \) represents the parameters of the closed loop. A state value in future can be computed by multiplying the matrices \( D \), so \( x_{k+N} = D^N x_k \). It is now easy to see that the eigenvalues of \( D \) give an indication of the stability. If they are larger than one the system is unstable and if they are smaller than one the system is stable. The closer the eigenvalues of \( D \) are to zero, the faster the state value will approach zero. If we include the noise \( v_k \) then this will disturb the state transition. The eigenvalues of \( D \) determine how many time steps it will take before the effect of the noise at time step \( k \) can be neglected.

For an optimality based design method first an evaluation criterion has to be chosen that indicates the desired behavior of the system. This criterion usually represents the cost of being at a certain state or applying certain control actions. Since the system is dynamic also the future consequences of the action on the costs should be taken into

\(^2\)For simplicity we ignore the noise.
1.1. CONTROL

account. This makes the optimization less trivial. Different methods are available to find the best controller:

- **Linear Quadratic Regulation**
  In case the system is linear and the costs are given by a quadratic function of the state and control action. It can be shown that the optimal control action is a linear function of the state \([11][3]\). The total future costs for each state forms a quadratic function of the state value. The parameters of this function can be found by solving an algebraic equation. The optimal linear function then follows from the system and the quadratic function.

- **Model Based Predictive Control**
  Using the model of the system, the future of the system can be predicted to give an indication of the future costs. The system does not have to be linear and the costs do not have to be given by a quadratic function. The optimal control sequence for a finite horizon is computed. The first action from this sequence is applied to the system. In the next time step the whole procedure is repeated, making this a computationally expensive method. For linear systems and constraints there are several optimization methods available to efficiently compute the optimal action at each time step \([58][42]\).

- **Dynamic Programming**
  The state and action space can be quantified and then Dynamic Programming can be used \([10][11][57]\). Starting at a finite point in future, the optimal control sequence is computed backwards using the model of the system.

Dynamic Programming will be used in chapter 2 to introduce reinforcement learning. In chapter 3 we will use Linear Quadratic Regulation.

The linear system in (1.2) was introduced to simplify the explanation. Controller design methods are best understood when applied to linear system. The main reason for this is that models of linear systems are mathematically convenient and the properties of these systems are well defined. Since not all systems are linear controller design approaches for nonlinear systems as in (1.1) were developed. For an overview of these methods see \([37]\). An alternative approach is to formalize the representation of the nonlinear system in such a way that many well-defined properties of linear systems can be used to describe the properties of nonlinear systems \([54]\). These properties do not have to be globally valid, so the state values for which they hold should also be given.

### 1.1.2 Unknown systems

The control design methods described above can only be applied if the model of the system is known exactly. This is generally not the case, for example when not the entire system is modeled or the specific values of the parameters are not known. So system identification methods were developed to approximate the model of the system based on measurements.
The measurements are generated by applying control actions to the system. The control actions have to be chosen such that they sufficiently excite the system. In that case the properties of the state transition function can be estimated from the data set containing all measurements and control actions. There are different possibilities available for identifying the model of the system:

- **Linear models**
  If the system is assumed to be linear the function describing the state transition is given. Only the parameters of the model have to be estimated. The identification methods for linear systems are well understood [46][22]. The parameters are usually estimated using a (recursive) linear least squares estimation method.

- **Local models**
  One global model can be build up using many smaller local models. These local models are only valid in a restricted part of the state space or their contribution to the output of the global model is restricted. In Gain Scheduling (GS) [3] the output of the global model is computed based on only one local model. The value of the state determines which local model to use.

  In locally weighted representations [4][5] all local models can contribute to the output of the global model. The contribution of the outputs of the local models depends on the weighting. The weighting depends on the value of the state. The local models can be linear. By combining more than one linear model a nonlinear function can be approximated by the global model.

  An alternative approach is to use Fuzzy logic, where reasoning determines the local models to use. In case of overlapping membership functions, the combination of local models is used to compute the control action [73]. If the system is unknown, the membership functions are hard to determine. In that case it is possible to use adaptive membership functions, that are trained using data from the system.

- **Nonlinear models**
  When the system is nonlinear it is not possible to use a linear model, if the deviation of this model from the system is too large. This means the parameters of a nonlinear model have to be found. If the function class of the system is unknown, general function approximator like neural networks can be used [53][36][86][63]. The parameters of the model are found by using a supervised learning method to train the networks. This means that quadratic error between the real next state value and that given by the model is minimized.

  Once the model of the system is found it can be used to design a controller. It is also possible to do this online, in which this is called indirect adaptive control. This can be applied when the system is varying in time or when the model forms a simplified local description of the system. The identification is used to find the parameters of the model and this model is used to tune the parameters of the controller.
If we adapt the controller it does not necessarily have to be based on the estimated model of the system. It is also possible that the design specification can be estimated directly from the measurements. This is called direct adaptive control. In this case different possibilities exist:

- **Self tuning regulator**
  The self tuning regulator as described in [3] directly estimates the parameters required to adjust the feedback. This can be based on the difference between the state value and some reference signal.

- **Model modification**
  It is possible in case of a nonlinear system that an approximator is used to change the current model into a model that makes control easier. An example is using a feed forward network to model the inverse kinematics of a robot as in [64]. Another example is feedback linearization, where a nonlinear feedback function is adapted to compensate for the nonlinearity in the system [78].

### 1.2 Reinforcement Learning

Reinforcement Learning [40][12][71][30] from a Machine Learning [48] point of view is a collection of algorithms that can be used to optimize a control task. Initially it was presented as a “trial and error” method to improve the interaction with a dynamical system [9]. Later it has been established that it can also be regarded as a heuristic kind of Dynamic Programming (DP) [80][83][8]. The objective is to find a policy, a function that maps the states of the system to control actions, that optimizes a performance criterion.

Compared with control, we can say that the policy represents the feedback function. In RL, the feedback function is optimized during the interaction using an evaluation from the system. Therefore we can also regard RL as a form of adaptive optimal control. During the interaction each state transition is being evaluated, resulting in a scalar reinforcement. The performance criterion to optimize is usually given by the expected sum of all reinforcements. This criterion has to be maximized if the reinforcements represent rewards and minimized if they represent costs.

The value function represents the expected sum of future reinforcements for each state. In RL the optimization is performed by first estimating the value function. The value function can be approximated by using Temporal Difference learning. In this learning rule the approximated value of a state is updated based on the difference with the approximated value of the next state. This difference should agree with the reinforcement received during that state transition.

When the (approximated) future performance is known for each state, then given the present state it is possible to select the “preferred” next state. But this still does not indicate what action should be taken. For this a model of the system is required. With this model, the action that has the highest probability of bringing the system in the “preferred” next state can be selected. Because such a model is not always available, model free RL
techniques were developed. Q-Learning is model-free RL [80][81]. The idea is that the sum of future reinforcements can be approximated as a function of the state and the action. This function is called the Q-function and it has a value for each state and action combination. The Q-function can be used to select that action for which the value of the Q-function is optimal, given the present state. This model-free optimization is only possible because the optimization is performed on-line by interacting with the system. Therefore we can see Q-Learning as a direct adaptive control scheme.

Approximations are made for the visited states and actions taken in these states. The use of the approximation to choose the best action can only rely on the actions that are tried often enough in these states. A deterministic policy, which always maps the state to the same action, cannot be used to generate the data for the approximation. So the actions taken during the interaction should not only depend on the existing policy, but should also depend on a random “trial” process. This process is referred to as the exploration. Exploration is of utmost importance because it determines the search space, in which the optimal policy is searched for. If the search space is large enough then it can be proven that RL converges to the optimal solution [68][25][24][39]. But these proofs rely on perfect backups of visited states and applied actions in look-up-tables. So guarantees for convergence to the optimal solution can only be given, when there are a discrete number of states and actions.

That the convergence proofs rely on a discrete state and action space does not mean that RL cannot be applied to systems with a continuous state and action space as in (1.1). In [65] RL is used to obtain a controller for a manufacturing process for thermoplastic structures, followed by a molding process and chip fabrication in [66]. In [2] and [59] a thermostat controller is obtained. A controller for satellite positioning is found in [61]. The use of RL for chemical plants and reactors are described in [47][59] and [60]. In [26] RL is used to control a “fuzzy” ball-and-beam system. The approaches used for these application were primarily based on heuristics, so there hardly is any theoretic understanding of the closed loop performance of these approaches.

1.3 Problem Statement

The reinforcement learning approach has been successfully applied as a control design approach for finding controllers. In the previous section we gave some examples. These approaches were all, except for [26], performed on simulated systems. In most cases the resulting controllers were used afterwards to control the real system.

The advantage of the reinforcement learning approach is that it does not require any information about the system. Only the measurements, generated by “trial and error”, are used to estimate the performance. When these approaches are applied in simulation, this advantage is not exploited. The perfect model of the system is still required. Still the successful results suggest that RL can be used to find good controllers.

There are two reasons why most RL approaches are applied in simulation and not directly on the real system. The first reason is that RL learning can be very slow. This is
because the process of interacting with the system has to be repeated several times. The duration of one time step for the real system is given by the system itself and for a slow process like a chemical plant learning will take a very long time. In simulation the duration of one time step is given by the time it takes to simulate one time step. If learning takes too much time a faster computer can be used to speed up the learning process.

A more serious problem with RL applied to real systems is that most algorithms are based on heuristics. This implies that the outcome will not always be completely understood. For a real system it is very important that properties, like stability, can be guaranteed to hold. Instability may cause the damage of the system. This means that without a better understanding, RL algorithms cannot be applied directly to a real system so that its advantage cannot be exploited.

The goal in this thesis is to make reinforcement learning more applicable as controller design approach for real systems. This means we want to derive methods that:

- are able to deal with continuous state space problems. Most reinforcement learning approaches are based on systems with discrete state and action space configurations. However, controllers for real systems often have continuous state values as input and continuous actions as output.

- are able to deal with nonlinear systems.

- do not require too high disturbances on the system. The excitation of the system involves adding a random component to the control action. For real systems this is not desirable, so we want to minimize the amount of excitation required.

- do not require too much data. The data is obtained by controlling the system, which means that the sample time of the system determines how long it takes to generate one data point. For a large sample time it means that generating a lot of data takes a long time during which the system can not be used.

- provide ways to interpret the results. This means that we want to have an indication of the quality of the resulting controller without testing it on the real system. Also, we want to be able to see how we can change the settings to achieve better results.

Methods that do not have any of these properties will not be very useful. Therefore all the methods we derive will have some of these properties.

### 1.4 Overview of this Thesis

This thesis is about RL and therefore we will start with a more in-depth introduction of RL in chapter 2. Since most theoretic results in RL address configurations with discrete state and action spaces, these configurations will be used to explain the basic idea behind RL. Then we will describe RL methods for continuous state space configurations, because
these are the systems we want to control. After that we refine our problem statement and choose a direction for our investigation.

In chapter 3 we choose the Linear Quadratic Regulation (LQR) framework as optimal control task. We introduce LQRQL as Q-learning approach to obtain the feedback for this framework. We compare the results with an indirect approach that first estimates the parameters of the system and uses this to derive the feedback. By introducing the exploration characteristics we reveal the influence of the amount of exploration on the resulting feedbacks of both methods. Based on this we can prove whether the resulting feedback will be an improvement, given the amount of exploration used.

In chapter 4 we investigate the consequences of applying LQRQL on nonlinear systems. We will indicate a possible shortcoming and introduce Extended LQRQL as a solution. This will result in a linear feedback with an additional constant. This approach and the two approaches from chapter 3 will be tested on a nonlinear system in simulation and on the real system.

In chapter 5 we will introduce Neural Q-Learning. The difference with the approach in [65] is that it only uses one neural network. The controller is derived in the same way as in LQRQL. It can be used to obtain a linear and a nonlinear feedback function. The conclusion and suggestions for future work are described in chapter 6.