

# Continuous State Space Q-Learning for Control of Nonlinear Systems



# Continuous State Space Q-Learning for Control of Nonlinear Systems

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor  
aan de Universiteit van Amsterdam,  
op gezag van de Rector Magnificus  
prof. dr. J.J.M. Franse  
ten overstaan van een door het college voor promoties ingestelde  
commissie, in het openbaar te verdedigen in de Aula der Universiteit  
op woensdag 21 februari 2001, te 12:00 uur

door

STEPHANUS HENDRIKUS GERHARDUS TEN HAGEN

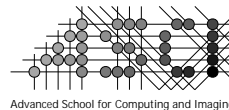
geboren te Neede

Promotor: Prof. dr. ir. F.C.A. Groen  
Co-promotor: Dr. ir. B.J.A. Kröse

Commissie: Prof. dr. H.B. Verbruggen  
Prof. dr. P. Adriaans  
Prof. dr. J.N. Kok  
Dr. M.A. Wiering

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

This work was part of STW project AIF.3595: “*Robust Control of Nonlinear Systems using Neural Networks*”. It was carried out in graduate school ASCI.



Copyright © 2001 by Stephan H.G. ten Hagen. All rights reserved.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Control . . . . .	2
1.1.1	Designing the state feedback controller . . . . .	3
1.1.2	Unknown systems . . . . .	5
1.2	Reinforcement Learning . . . . .	7
1.3	Problem Statement . . . . .	8
1.4	Overview of this Thesis . . . . .	9
<b>2</b>	<b>Reinforcement Learning</b>	<b>11</b>
2.1	A Discrete Deterministic Optimal Control Task . . . . .	11
2.1.1	The problem . . . . .	11
2.1.2	The solution . . . . .	13
2.2	The Stochastic Optimization Tasks . . . . .	13
2.2.1	The Markov Decision Process . . . . .	13
2.2.2	Model based solutions . . . . .	14
2.2.3	Reinforcement Learning solutions . . . . .	16
2.2.4	Advanced topics . . . . .	20
2.2.5	Summary . . . . .	21
2.3	RL for Continuous State Spaces . . . . .	22
2.3.1	Continuous state space representations . . . . .	22
2.3.2	Learning in continuous domains . . . . .	24
2.3.3	Summary . . . . .	26
2.4	Discussion . . . . .	27
<b>3</b>	<b>LQR using Q-Learning</b>	<b>29</b>
3.1	Introduction . . . . .	29
3.2	LQR with an Unknown System . . . . .	30
3.2.1	Linear Quadratic Regulation . . . . .	30
3.2.2	System Identification . . . . .	31
3.2.3	The Q-function . . . . .	32
3.2.4	Q-Learning . . . . .	33
3.2.5	The Performance Measure . . . . .	35
3.2.6	Overview . . . . .	36

3.3	The Influence of Exploration . . . . .	37
3.3.1	The estimation reformulated . . . . .	37
3.3.2	The System Identification approach . . . . .	38
3.3.3	The LQRQL approach . . . . .	42
3.3.4	The Exploration Characteristics . . . . .	45
3.4	Simulation Experiments . . . . .	47
3.4.1	Setup . . . . .	47
3.4.2	Exploration Characteristic . . . . .	47
3.5	Discussion . . . . .	49
3.6	Conclusion . . . . .	50
<b>4</b>	<b>LQRQL for Nonlinear Systems</b>	<b>51</b>
4.1	Introduction . . . . .	51
4.2	Nonlinearities . . . . .	51
4.2.1	The nonlinear system . . . . .	51
4.2.2	Nonlinear approaches . . . . .	53
4.2.3	Summary . . . . .	56
4.3	The Extended LQRQL Approach . . . . .	56
4.3.1	SI and LQRQL for nonlinear systems . . . . .	56
4.3.2	The extension to LQRQL . . . . .	57
4.3.3	Estimating the new quadratic Q-function . . . . .	58
4.4	Exploration Characteristic for Extended LQRQL . . . . .	59
4.5	Simulation Experiments with a Nonlinear System . . . . .	63
4.5.1	The nonlinear system . . . . .	63
4.5.2	Experiment with a nonzero average $\mathbf{w}$ . . . . .	66
4.5.3	Experiments for different average $\mathbf{w}$ . . . . .	68
4.6	Experiments on a Real Nonlinear System . . . . .	70
4.6.1	Introduction . . . . .	70
4.6.2	The experiments . . . . .	70
4.7	Discussion . . . . .	73
4.8	Conclusions . . . . .	74
<b>5</b>	<b>Neural Q-Learning using LQRQL</b>	<b>75</b>
5.1	Introduction . . . . .	75
5.2	Neural Nonlinear Q-functions . . . . .	76
5.3	Neural LQRQL . . . . .	78
5.3.1	Deriving the feedback function . . . . .	79
5.3.2	Discussion . . . . .	81
5.4	Training the Network . . . . .	82
5.5	Simulation Experiments with a Nonlinear System . . . . .	83
5.5.1	Introduction . . . . .	83
5.5.2	The experimental procedure . . . . .	85
5.5.3	The performance of the global feedback functions . . . . .	87

---

5.5.4	Training with a larger train set . . . . .	88
5.6	Experiment on a Real Nonlinear System . . . . .	90
5.7	Discussion . . . . .	92
5.8	Conclusions . . . . .	93
<b>6</b>	<b>Conclusions and Future work</b>	<b>95</b>
6.1	LQR and Q-Learning . . . . .	96
6.1.1	Future work . . . . .	97
6.2	Extended LQRQL . . . . .	97
6.2.1	Future work . . . . .	98
6.3	Neural Q-Learning . . . . .	98
6.3.1	Future Work . . . . .	99
6.4	General Conclusion . . . . .	99
<b>A</b>	<b>The Least Squares Estimation</b>	<b>101</b>
A.1	The QR-Decomposition . . . . .	101
A.2	The Least Squares Solution . . . . .	101
A.3	The SI Estimation . . . . .	103
<b>B</b>	<b>The Mobile Robot</b>	<b>105</b>
B.1	The robot . . . . .	105
B.2	The model of the robot . . . . .	106
<b>C</b>	<b>Notation and Symbols</b>	<b>107</b>
C.1	Notation . . . . .	107
C.2	Symbols . . . . .	108
	<b>Bibliography</b>	<b>110</b>
	<b>Summary</b>	<b>117</b>
	<b>Samenvatting</b>	<b>118</b>
	<b>Acknowledgments</b>	<b>120</b>



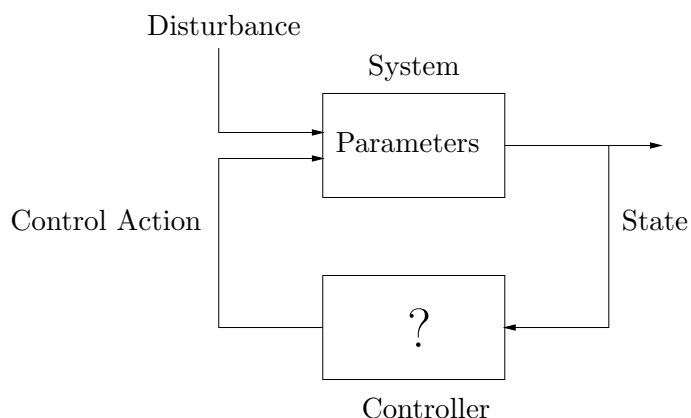
# Chapter 1

## 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



**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 an 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  $\mathbf{x} \in \mathbb{R}^{n_x}$ , the control action as vector  $\mathbf{u} \in \mathbb{R}^{n_u}$  and the noise as vector  $\mathbf{v} \in \mathbb{R}^{n_v}$ , the state transition is given by:<sup>1</sup>

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k). \quad (1.1)$$

This is the model of the system. The  $k$  in (1.1) represents the time step. The  $\mathbf{f}$  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  $\mathbf{f}$ .

In (1.1) the vector  $\mathbf{v}$  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  $\mathbf{v}$  can also be used to take into account the consequences of the unmodeled dynamics of the system. In the design of the controller  $\mathbf{v}$  is regarded as a stochastic variable.

Note that the model in (1.1) represents a Markov process. This means that the next state  $\mathbf{x}_{k+1}$  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  $\mathbf{x}$ ,  $\mathbf{u}$  and  $\mathbf{v}$  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  $\mathbf{u} = \mathbf{g}(\mathbf{x})$ . With these assumptions, designing the controller becomes a matter of finding the appropriate  $\mathbf{g}$ . The function  $\mathbf{g}$  will be called the *feedback*.

### 1.1.1 Designing the state feedback controller

The goal of the design is to find the best  $\mathbf{g}$  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

---

<sup>1</sup>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  $\mathbf{f}$  represents a linear function of the state, the control action and the noise:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{v}_k, \quad (1.2)$$

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:

$$\mathbf{u} = L\mathbf{x}, \quad (1.3)$$

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<sup>2</sup>  $\mathbf{x}_{k+1} = (A + BL)\mathbf{x}_k = D\mathbf{x}_k$ , where  $D$  represents the parameters of the closed loop. A state value in future can be computed by multiplying the matrices  $D$ , so  $\mathbf{x}_{k+N} = D^N\mathbf{x}_k$ . 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  $\mathbf{v}_k$  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

---

<sup>2</sup>For simplicity we ignore the noise.

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 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 necessary 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 system 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 system 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.
- provides 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.

# Chapter 2

## Reinforcement Learning

In the previous chapter we described Reinforcement Learning (RL). In this chapter we will give an introduction to RL in which we explain the basic RL concepts necessary to understand the remainder of this thesis. For a more complete overview of RL see [40][12][71].

Most theoretical results on RL apply to systems with discrete state and action spaces. Therefore we will start our introduction based on these systems. First we describe the optimal control task for a deterministic system and present a systematic procedure to compute the optimal policy. Then we introduce the Markov Decision Process (MDP) and describe the classical solution approaches that use the model of the system. In RL the optimization is based on the interaction with the system. We will describe two important RL algorithms; Temporal Difference learning and Q-Learning.

We want to use RL for systems as described in the previous chapter, which are systems with continuous state and action spaces. One way of using RL methods in the continuous state and action space task is changing the problem into a discrete state and action problem by quantization of the state and action spaces. Then the original discrete algorithms can be used. The other way is to modify the algorithms so that they can deal with general function approximators operating in the continuous domain. We will focus on the last. We will conclude with a discussion in which we refine our problem statement.

### 2.1 A Discrete Deterministic Optimal Control Task

To introduce the basics of an optimal control task we will restrict ourselves to a deterministic discrete system.

#### 2.1.1 The problem

A discrete deterministic optimal control task consists of:<sup>1</sup>

---

<sup>1</sup>We denote the discrete states and actions by  $s$  and  $a$  to make a clear distinction between the continuous states  $\mathbf{x}$  and actions  $\mathbf{u}$  described in chapter 1.

- A finite set of states  $\{s^1, s^2, \dots, s^{n_s}\}$   
The indices indicate the labels of the states of the system.
- A set of actions  $\{a^1, a^2, \dots, a^{n_a}\}$   
The set of actions can depend on the state, because it is possible that not all actions are possible in all states.
- A dynamic system  
The state transition of the system changes the current state of the system to a possible other state. To denote the state at a certain time step we will use the time as an index. So  $s_k$  is a particular element from the set of states at time step  $k$ . For the deterministic system the state transition maps the present state and action to the next state.
- Reinforcements  $r_k$   
These indicate the received reinforcement at time step  $k$ . There does not have to be a dependency on time, in general it depends on the state transition and action that takes place at time step  $k$ .
- The criterion  
The criterion indicates the desired behavior. It describes how all reinforcements at different time steps are combined to give a scalar indication of the performance. This can be the sum over all reinforcements that have to be either minimized or maximized, depending on whether the reinforcements are interpreted as costs or rewards. Another possible criterion is based on the average of reinforcements [67].

As the state transitions are deterministic, an action taken in one state always results in the same next state. The *policy*  $\pi$  is the function that maps the states to the action, so that action  $a = \pi(s)$  is taken in state  $s$ . Given a policy for a deterministic system, the entire future sequence of states and actions is determined.

We have to specify our criterion before we can solve this optimization task. We take as criterion the minimization of the cost to go to a certain goal state. For a given policy  $\pi$  the future is already determined, so we can compute the total costs to the goal state for each state. We will call this the value function<sup>2</sup>  $V^\pi$ , and  $V^\pi(s)$  is the value of state  $s$

$$V^\pi(s) = \sum_{i=k}^N r_i, \quad (2.1)$$

where  $s = s_k$  and  $N$  is the final time step when the goal state is reached. So now the optimization task is to find an *optimal policy*<sup>3</sup>  $\pi^*$  for which the values of all states are minimal. Note that the corresponding optimal value function  $V^* = V^{\pi^*}$  is unique, but the optimal policy does not have to be unique.

<sup>2</sup>If the number of discrete states is finite, the value function can be regarded as a vector  $V^\pi \in \mathbb{R}^{n_s}$ . Then  $V^\pi(s)$  is the element from vector  $V^\pi$  corresponding to state  $s$ . A function/vector like this is often called a look-up-table.

<sup>3</sup>We will use \* to denote optimal solutions.

### 2.1.2 The solution

Computing the optimal value function can help determining the optimal policy. A systematic procedure is to compute the optimal value function backwards, starting in the goal state. For all states that can reach the goal state, store the minimal costs as the value of these states. Store also the actions that lead to these minimal costs, because they will form the optimal policy. The policy we get is the optimal policy for going in one step to the goal state.

Now we consider all states that can reach the goal state in two steps. For each state select the action for which the received cost *plus* the stored value of the next state is minimal. We can store this as the new value for each state, but first we have to check whether this state already has a value stored. This is when the goal state can also be reached in one step for this state. Only if the new value is lower than the already stored value, we change the stored value and action. This gives the optimal policy for going in two steps to the goal state. This procedure can be repeated until each state that can reach the goal state has an optimal action stored. Then we have the optimal policy.

The underlying idea of this procedure is that for the optimal value function, the following relation should hold:

$$V^*(s) = \min_a \{r_k + V^*(s')\}. \quad (2.2)$$

Here the  $s$  is a state for which any  $s'$  can be reached in one time step. Here  $r_k$  represents the cost received at time step  $k$  when action  $a$  is applied and the state changes from  $s$  to  $s'$ . For all possible actions in state  $s$  the left hand side is computed and the action  $a$  is determined for which this is minimal. The minimal value is stored as  $V^*(s)$ .

The procedure described is called Dynamic Programming (DP) [10] and a more general form of (2.2) is called the Bellman equation. It defines optimal actions as actions for which the minimal value of the two successive states only differ in the cost received during that time step. As illustrated by DP, the Bellman equation can also be used to find algorithms to derive the optimal actions.

## 2.2 The Stochastic Optimization Tasks

The solution to the deterministic optimal control task described before is quite straightforward. But this is just a restricted class of optimal control tasks. A more general class of optimization task is given by Markov Decision processes (MDP). This will be the framework in which we will introduce RL algorithms, but first we will describe some model based solutions.

### 2.2.1 The Markov Decision Process

The MDP consist of the same elements as the deterministic optimal control task. The difference is that the state transitions and reinforcements no longer have to be deterministic.

The state transitions of the process are given by all transition probabilities for going from some state  $s$  to some other state  $s'$  when action  $a$  is taken:

$$P_{ss'}^a = \Pr \{s_k = s' \mid s_k = s, a_k = a\}. \quad (2.3)$$

All the  $n_s \times n_s \times n_a$  probabilities together form the model of the system. The deterministic system is just a special case of (2.3), for which all values of  $P_{ss'}^a$  are either one or zero. The process described in (2.3) is a Markov process because the state transitions are conditionally independent of the past. This is a necessary property to express the value function as a function of the current state alone.

The value function for the MDP can be defined similar to (2.1). To take into account the stochastic state transitions according to (2.3) the expectation value has to be taken over the sum:

$$V^\pi(s) = E_\pi \left\{ \sum_{i=k}^N r_i \mid s_k = s \right\}. \quad (2.4)$$

The subscript  $\pi$  of the policy is used to indicate that actions in the future are always selected according to policy  $\pi$ .

Because the state transitions are probabilistic, the final time step  $N$  (at which the goal state is reached) can vary a lot. This is because one state transition can lead to different next states. This means that all state transition after that can be different, and that it may take a different number of time steps to reach the goal state. It is even possible that  $N$  becomes infinite.

The consequence is that the sum in the expectation value of (2.4) can have many different values, so that the variance of the underlying probability density function is very high. A discount factor  $\gamma \in [0, 1]$  can be introduced to weigh future reinforcements. This will reduce the variance and make the value function easier to approximate. So a more general definition of the value function for a stochastic system becomes:

$$V^\pi(s) = E_\pi \left\{ \sum_{i=k}^N \gamma^{i-k} r_i \mid s_k = s \right\}. \quad (2.5)$$

Note that there can be another reason to include a discount factor. This is the situation where all reinforcements are zero except for the goal state. In this situation (2.4) reduces to  $V^\pi(s) = r_N$ , so that all policies that eventually will reach the goal state will be equally good. A discount factor  $\gamma < 1$  in (2.5) will make states closer to the goal state have higher values. Then the optimal policy is the one that will reach the goal state in the smallest number of steps.

## 2.2.2 Model based solutions

We will now describe some solution methods that are based on computing the value function using the model of the system. So all probabilities (2.3) have to be known. If they are not known, these probabilities have to be estimated by interacting with the system. Once the model is available, the system itself will no longer be used to compute the value function.

The value function is computed to solve the stochastic optimal control task. A solution method we already described is DP, that uses backwards induction from a final time step in future. For the stochastic optimization task, a more general form of the Bellman equation (2.2) has to be used:

$$V^*(s) = \sum_{s'} P_{ss'}^{\pi^*} (R_{ss'}^{\pi^*} + \gamma V^*(s')). \quad (2.6)$$

The  $P_{ss'}^{\pi^*}$  is a matrix with state transition probabilities from  $s$  to  $s'$  when taking actions according to the optimal policy  $\pi^*$ . The vector  $R_{ss'}^{\pi^*}$  represents the expected reinforcements assigned to the state transitions when the optimal action is taken

$$R_{ss'}^{\pi^*} = E \{ r_k \mid s_k = s, s_{k+1} = s', a_k = \pi^*(s_k) \}. \quad (2.7)$$

The main difference between (2.6) and (2.2) is that now all possible next states are taken into account, weighed with the probabilities of their occurrence. So the optimal policy is the policy for which the expected sum of future reinforcements is minimal.

We can define a  $n_s \times n_s$  matrix  $P^{\pi^*}$  with state transition probabilities when policy  $\pi^*$  is used. Here the rows indicate all the present states and the columns all the next states. We can also define a vector  $R^{\pi^*}$  with the expected reinforcements for each state. Then (2.6) can be written more compact as  $V^* = P^{\pi^*} (R^{\pi^*} + \gamma V^*)$ . The compact notation of the Bellman equation indicates that DP requires that all state transitions with nonzero probabilities have to be taken into account. This makes it a very difficult and computationally expensive method.

## Policy Iteration

The Bellman equation (2.6) provides a condition that has to hold for the optimal value function. Based on this equation some iterative methods were developed to compute the optimal policy. One of them is *Policy Iteration* [11][57], which consists of two basic steps. The first step is the policy evaluation, in which for a given policy the value function is computed. The second step is the policy improvement step in which the new policy is computed. To compute the new policy, the *greedy* policy is determined. This is the policy for which in each state the action is chosen that will result in the next state with the lowest value.

The policy evaluation is an iteration algorithm based on (2.6), with the difference that the action is taken according to the policy that is being evaluated. The expectancy in the iteration can be calculated using the known transition probabilities  $P_{ss'}^{\pi_m(s)}$  and their expected reinforcements  $R_{ss'}^{\pi_m(s)}$ . Here  $\pi_m$  is the policy at iteration step  $m$ , that is being evaluated. Starting with an initial  $V_0$  the values for each state are updated according to

$$V_{l+1}(s) = \sum_{s'} P_{ss'}^{\pi_m} (R_{ss'}^{\pi_m} + \gamma V_l(s')), \quad (2.8)$$

where  $l$  indicates the policy evaluation step. Note that one policy evaluation step is completed when (2.8) is applied to *all* states.

Similar to DP, the policy evaluation can be regarded as computing the value function backwards. To see this, take the expected reinforcement *at* the finite time step  $N$  as the initial  $V_0$ . Then (2.8) makes  $V_1$  become the expected cost of the final state transition plus the discounted cost at the final state. The  $V_2$  becomes the future costs of the last two state transitions and this goes on for every iteration using (2.8). The  $V_N$  equals the  $V^\pi$  in (2.5), so that the correct value function is found.

The practical problem of the policy evaluation is that  $N$  can be unknown or infinite. Fortunately the difference between  $V_{l+1}$  and  $V_l$  will decrease as  $l$  increases. The effect is that after a certain value of  $l$  the greedy policy corresponding to the value function in (2.8) will not change. Then the policy evaluation can stop and the policy can be improved by taking the greedy policy. The greedy policy is given by

$$\pi'_m(s) = \operatorname{argmin}_a \left\{ \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V_l(s')) \right\}, \quad (2.9)$$

where  $\pi'_m$  is the greedy policy. The  $V_l$  in the right hand side of (2.9) indicates that the greedy policy is computed after  $l$  evaluation steps. The number of evaluation steps is increased until  $\pi'_m$  no longer changes. Then the policy is improved by taking as new policy  $\pi_{m+1} = \pi'_m$ . Because the number of policies is finite, the optimal policy  $\pi^*$  will be found after a final number of policy iteration steps.

## Value Iteration

The difference between policy iteration and DP is that policy iteration does not use the optimal action during policy evaluation. It takes the actions according to the current policy. In DP only the actions that are optimal are considered. A combination of these two methods is called *value iteration*.

Value iteration is a one step approach in which the two policy iteration steps are combined. This is done by replacing  $\pi(s)$  in  $P_{ss'}^{\pi(s)}$  and  $R_{ss'}^{\pi(s)}$  by the greedy policy of  $V_m$ . So it still takes into account all possible state transitions, but now the policy is changed immediately into the greedy policy. Under the same conditions as policy iteration, value iteration can be proven to converge to the optimal policy.

Note that there still is a difference with DP. Value Iteration does not use the optimal action, but actions that are assumed to be optimal. This does not have to be true. Especially in the beginning of value iteration, it is very unlikely that the approximated value function is correct. Therefore it is also very likely that the actions used by Value Iteration are not optimal.

### 2.2.3 Reinforcement Learning solutions

The model based solution methods *only* use the model of the system to compute the optimal policy. The system itself is not used once the model is available. This is the main difference with reinforcement learning solutions. The reinforcement learning solution methods use *interactions* with the system to optimize the policy. To enable the interaction with the

system, an initial policy has to be available. The system starts in the initial state and the interaction will go on until the final time step  $N$ . This process can be repeated several times.

### Temporal Difference learning

The policy  $\pi$  is available and the data obtained by the interaction with the system is used to evaluate the policy. With the interactions, the value function is only updated for the present state  $s = s_k$  using only the actual next state  $s' = s_{k+1}$ . Also the received reinforcement  $r_k$  is used and not the expected reinforcement  $R_{s,s'}^\pi$ . This suggests an update according to:

$$V_{l+1}(s_k) = r_k + \gamma V_l(s_{k+1}). \quad (2.10)$$

Clearly this is not the same as (2.8), because it is based on *one* state transition that just happens with a certain probability. In (2.8) the value function is updated for all states simultaneously using all possible next states.

The interaction with the system can be repeated several times by restarting the experiment after the goal state is reached. For one particular state, the next state and received reinforcement can be different. So the right hand side of (2.10) can have different values in repeated experiments. Computing the average of these values before updating the value function in (2.10), will result in the same update as (2.8) if the interactions are repeated infinite times.

A drawback of this approach is that all right hand sides of (2.10) for each state have to be stored. It is also possible to incrementally combine the right hand side of (2.10) with an existing approximation of  $V_l$ . This leads to an update like

$$V_{l+1}(s_k) = (1 - \alpha_l)V_l(s_k) + \alpha_l(r_k + \gamma V_l(s_{k+1})). \quad (2.11)$$

The  $\alpha_l \in [0, 1]$  is the learning rate. For a correct approximation of the value function, the learning rate should decrease. Therefore the learning rate has the index  $l$ . If (2.11) is rewritten to

$$V_{l+1}(s_k) = V_l(s_k) + \alpha_l(r_k + \gamma V_l(s_{k+1}) - V_l(s_k)). \quad (2.12)$$

we see that  $V_l(s_k)$  is changed proportional to the difference between the value of the current state and what it should be according to the received cost and the value of the next state. This difference is called the *Temporal Difference* (TD) and the update (2.12) is called temporal difference learning [68].

### Exploration

There is one problem with approximating the value function based on visited states alone. It is possible that some states are never visited, so that no value can be computed for these states. So in spite of repeating the interaction, it is still possible that (2.12) never equals (2.8). To prevent this an additional random trial process is included, that is referred

to as the *exploration*. The exploration has the same function as the excitation in system identification, described in chapter 1.

The exploration means that sometimes a different action is tried than the action according to the policy. There are different strategies to explore. In [76] a distinction is made between directed and undirected exploration. In case of undirected exploration, the tried actions are truly random and are selected based on a predefined scheme.

Undirected exploration does not take information about the system into account but can use experiences gathered during previous interactions. One popular undirected exploration strategy is given by the  $\epsilon$ -greedy policy<sup>4</sup> [70]. With a probability  $1 - \epsilon$  the action according to the greedy policy is taken. With a probability  $\epsilon$  a non-greedy action is tried. In this way the parameter  $\epsilon$  can be used to specify the amount of exploration. Usually the initial value of  $\epsilon$  is taken high, so that many actions are tried. With each policy improvement step also the value of  $\epsilon$  is decreased. The effect is that when the policy becomes better, the exploration focuses more and more around the improved policies. The result is that a fewer number of interactions with the system is required.

Exploration can also help to converge faster to the optimal policy. This is when an action is tried that leads to a state with a lower value, that would not be reached when the action according to the policy was taken. The consequence is that the value of the state where the action was tried is decreased. This increases the probability that the improved policy visits that state. This can be seen as finding a shortcut because of the exploration.

## Q-Learning

The value function for a policy  $\pi$  can be approximated without a model of the system. However, computing the greedy policy using (2.9) still requires  $P_{ss'}^a$  and  $R_{ss'}^a$ . A model free RL method that does not use  $P_{ss'}^a$  and  $R_{ss'}^a$  was introduced. It was called Q-Learning [80][81]. It is based on the idea to estimate the value for each state and action combination.

The Bellman equation in (2.6) uses  $P_{ss'}^{\pi^*}$  and  $R_{ss'}^{\pi^*}$ , so it is only valid if in each state the optimal action  $\pi^*(s)$  is taken. It is also possible to compute the right hand side of (2.6) for each possible action using  $P_{ss'}^a$  and  $R_{ss'}^a$ . This gives the value for each state and action combination, when all actions in future are optimal. Define

$$Q^*(s, a) = \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s)) \quad (2.13)$$

as the optimal Q-function. For the optimal actions this agrees with the Bellman equation. For all other actions the value will be higher, so:

$$V^*(s) = \min_a Q^*(s, a). \quad (2.14)$$

For each state the optimal action is given by:

$$\pi^*(s) = \operatorname{argmin}_a Q^*(s, a). \quad (2.15)$$

---

<sup>4</sup>This is also known under the name pseudo-stochastic or max-random or Utility-drawn-distribution.

This implies that the optimal Q-function alone is sufficient to compute the optimal policy.

The Q-function for policy  $\pi$  can be defined in the same way as (2.5):

$$Q^\pi(s, a) = E_\pi \left\{ \sum_{i=k}^N \gamma^{i-k} r_{i+1} \mid s_k = s, a_k = a \right\} \quad (2.16)$$

Now temporal difference learning can be used to approximate  $Q^\pi(s, a)$ .

The temporal difference update in (2.12) uses the value for the present state  $s_k$  and the next state  $s_{k+1}$ . For the Q-function the update should use the present state-action combination and the next state-action combination. Here we have to be careful, because (2.16) requires that all future actions are taken according to policy  $\pi$ . This means that the next action  $a_{k+1}$  has to be  $\pi(s_{k+1})$ . The current action can be  $\pi(s_k)$ , but it can also be another action that is being explored. So the temporal difference update for the Q-function becomes

$$Q_{l+1}(s_k, a_k) = Q_l(s_k, a_k) + \alpha_l (r_k + \gamma Q_l(s_{k+1}, \pi(s_{k+1})) - Q_l(s_k, a_k)). \quad (2.17)$$

The greedy policy can be determined similar to (2.15) according to

$$\pi'(s) = \underset{a}{\operatorname{argmin}} Q_l(s, a). \quad (2.18)$$

This represents one policy iteration step. If the process is repeated, eventually the optimal policy will be found. Note that the original Q-learning in [80] was based on value iteration.

## Convergence

If the system can be represented by a MDP and for each state the value is estimated separately, the convergence of temporal difference learning can be proven [68][38][24][25]. Also, if the Q-value for each state-action combination is estimated separately, the convergence of Q-learning can be proven [80] [81]. The guaranteed convergence requires that the learning rate decreases and that the entire state action space is explored sufficiently.

The learning rate  $\alpha$  in (2.12) has to decrease in such a way that:

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \quad (2.19)$$

$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty \quad (2.20)$$

holds. The condition (2.20) is required to make sure that the update eventually will converge to a fixed solution. The condition (2.19) is required to make sure that the state space can be explored sufficiently. If  $\alpha$  decreases faster then it is possible that not all states are visited often enough, before the update of the value function becomes very small. This also requires that sufficient exploration is used, so that in theory all states are visited infinitely often.

## 2.2.4 Advanced topics

### The benefits of RL

Why should RL be used? An obvious reason for using RL would be if there is no model of the system available. However, such a model can be obtained by interaction with the system and estimating all probabilities. Then all the  $n_s \times n_s \times n_a$  probabilities have to be estimated. It is clear that the number of probabilities to estimate grows very fast when  $n_s$  increases.

When a model is available and used, all state transition probabilities are considered, so the computational complexity of one evaluation step is quadratic in the number of states and linear in the number of actions. The convergence rate of the iteration also depends on the number of states. There are  $n_s^{n_a}$  possible policies, so this is the maximum number of policy improvement steps required [57]. In general it is unknown how many policy evaluations steps are required.

The reinforcement learning methods can be beneficial when the state space becomes large. This is because updates are made only for the state transitions that actually take place. So the estimated values are primarily based on state transitions that occur with a high probability. The state transitions with low probabilities hardly occur and will not have a large influence on the estimated value function. In the worst case however, all states should be visited often enough. So in the worst case RL is not beneficial, but on average it will be able to find the optimal solution faster for problems with a large number of states.

### Increasing the speed

Although the RL algorithms require on average less computation than classical approaches, they still need to run the system very often to obtain sufficient data. Different solutions were proposed to speed up the learning algorithms:

- **Generalizing over time**

The temporal difference update (2.12) changes the value according to the difference between the value of the current state and what it should be according to the received reinforcement during the state transition and the value of the next state. This update is based on one state transition. Since the value represents the sum of future costs, the value can also be updated to agree with previous states and the reinforcements received since.

The update (2.12) can be written as  $V_{l+1}(s_k) = V_l(s_k) + \alpha_l \Delta V_k$ . The change  $\Delta V_k$  is in this case the temporal difference. To take also past states into account the update can be based on a sum of previous temporal differences. This is called TD( $\lambda$ ) learning [68], where the “discount factor”  $\lambda \in [0, 1]$  weighs the previous updates. The update of the value becomes:

$$\Delta V_k = (r_k + \gamma V_l(s_{k+1}) - V_l(s_k)) \tau(s_k), \quad (2.21)$$

where  $\tau$  represents the eligibility trace. For the present state the eligibility is updated according to  $\tau(s_k) := \lambda\gamma\tau(s_k) + 1$ , while for all other states  $s$  the eligibility is updated according to  $\tau(s) := \lambda\gamma\tau(s)$ . There are other possible updates for the eligibility [62], but the update in (2.12) always corresponds with TD(0). This can be further enhanced as in Truncated Temporal Difference (TTD) [23], where the updates are postponed until sufficient data is available. For Q-Learning it is also possible to use TD( $\lambda$ ) learning, resulting in Q( $\lambda$ ) learning [55]. Also this can be further enhanced by using fast online Q( $\lambda$ ) [84].

- **Using a model**

Although a model is not required for all RL techniques, a model can be used to speed up learning when it is available. If a model is not available it can be estimated simultaneously with the RL algorithm. This does not have to be a complete model with estimation of all state transition probabilities. It can also be trajectories stored from previous runs.

The model can be used to generate simulated experiments [45]. In DYNA [69] a similar approach is taken. Updates of the estimated value function are based on the real system or based on a simulation of the system. A different approach is to use the model to determine for which states the estimated values should be updated. In Prioritized Sweeping [49] a priority queue is maintained that indicates how promising the states are. Then the “important” states in the past are also updated based on the present state transition.

## 2.2.5 Summary

We introduced reinforcement learning for Markov decision processes with discrete state and action spaces. The goal is to optimize the mapping from states to control actions, called the policy. Because of the discrete state spaces, the value of the expected sum of future reinforcements can be stored for each state separately. These values can be computed off-line using the model of the system. When RL is used, these values are estimated based on the interaction with the system.

The estimated values for two successive states should agree with the reinforcement that is received during that state transition. Temporal Difference learning is based on this observation. The estimated value of the present states is updated in a way that it will agree more with the estimated value of the next state and the received reinforcement. To get a good approximation of the value for all states, all states have to be visited often enough. For this an additional random trial process, called exploration, is included in the interaction with the system. Also the whole interaction process itself should be repeated several times by restarting the system.

Once all values are estimated correctly the new policy can be determined. For this the actions are selected that have the highest probability of bringing the system in the most desirable next state. To compute these actions, the model of the system should be available. A different approach is Q-Learning. The expected sum of future reinforcements

is estimated for each state and action combination. Once this Q-function is estimated correctly the action can be selected based on the estimated values.

## 2.3 RL for Continuous State Spaces

### 2.3.1 Continuous state space representations

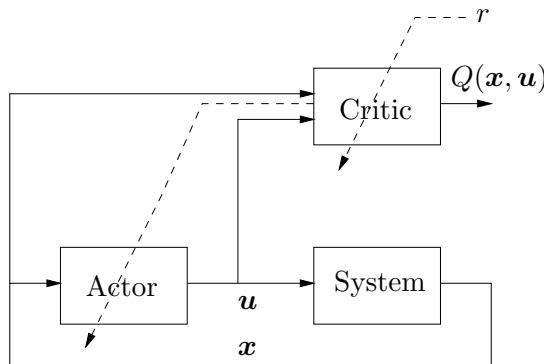
The RL approaches described in the previous section were based on estimating the value for each state or state-action combination. This is only possible if the number of states and possible actions is finite. So these RL approaches can only be used for systems with discrete state and action spaces. We want to use RL algorithms to find controllers for systems with continuous state and action spaces. These are the systems as described in chapter 1 with state  $\mathbf{x} \in \mathbb{R}^{n_x}$  and control action  $\mathbf{u} \in \mathbb{R}^{n_u}$ . This means that we cannot directly apply the algorithms to these systems. We will give two different approaches to apply RL to optimize the controllers for these systems.

#### State space quantization

A very obvious solution to get from a continuous state space to a discrete one is to quantize the state space. This is a form of state aggregation, where all states in a part of the state space are grouped to form one discrete state. In [9] the continuous state space of an inverted pendulum was partitioned into a finite number of discrete states. The advantage of this method is that the standard RL algorithms can be used, but there are also some drawbacks:

- If the continuous state space system is a Markov process, then it is possible that the discretized system is no longer Markov [50]. This means that the convergence proofs that apply for the standard RL algorithms no longer have to be valid.
- The solution found is probably suboptimal. Due to the quantization the set of possible policies is reduced considerably. The optimal policy for the continuous state space problem may not be in the set. In that case the optimal solution found for the discrete system may still perform very badly on the continuous system.

Because the choice of the quantization influences the result, algorithms were developed that use adaptive quantization. There are methods based on unsupervised learning, like k-Nearest Neighbor [28] or self organizing maps [43]. There is a method that uses triangularization of the state space based on data [51], which has been improved in [52]. There are divide and conquer methods, like the Parti-Game Algorithm [49], where large states are split when necessary. The advantage of these adaptive quantization methods is that they can result in a more optimal policy than the fixed quantization methods. On the other hand, when a large state is split the probability of being in one of the smaller states becomes smaller. The consequence is that the estimated values for these states become less reliable.



**Figure 2.1.** The Actor-Critic Configuration

### Function approximations

Function approximators are parameterized representations that can be used to represent a function. Often function approximators are used that can represent any function. In a RL context function approximators are often used in the *actor-critic* configuration as shown in figure 2.1. Two function approximators are used, one representing the policy called the actor and one representing the value function called the critic.

If we look at the critic we see that it has as input the state  $\mathbf{x}$  and the action  $\mathbf{u}$ .<sup>5</sup> This indicates that it is forming a Q-function rather than a value function. However, Q-learning using function approximators is hardly possible. This is because when the critic is a general function approximator, the computation of the action for which the approximated Q-function is minimal can be very hard. It is unlikely that an analytical function can be used for this, so the minimum has to be numerically approximated. In the actor-critic architecture the gradient of the critic is used to compute the update of the actor.

As function approximators often feed-forward neural networks were used. This approach has been successfully demonstrated in [74] and [75], which resulted in a program for Backgammon that plays at world champion level. Note also that [75] applies to a system with a finite number of states. The neural network was only used to generalize over the states.

The applications mentioned in section 1.2 were all based on function approximators. Feed forward neural networks were used in [65][59][60][26]. CMACs were used in [70][65][66]. Radial basis function were used in [1] in combination with a stabilizing controller.

In [14] examples are given where the use of RL with function approximators can fail. In [70] the same examples were used and the experimental setup was modified to make them work. Proofs of convergence for function approximation only exist for approximators linear in the weight when applied to MDPs [77][79]. For systems with a continuous state space there are no proofs when general function approximators are used. Also there are no general recipes to make the use of function approximators successful. It still might require

<sup>5</sup>Note that it is also possible to have a critic with only state  $\mathbf{x}$  as input.

some hand tuning in the settings of the experiment. Proofs of convergence for continuous state-action space problems do exist for the linear quadratic regularization task [82][17][44], but then no general function approximator is used but an approximator that is appropriate for this specific task.

### 2.3.2 Learning in continuous domains

The networks represent function approximators. We will describe how they can be trained. The critic is the function  $V(\boldsymbol{\xi}, \boldsymbol{w})$ , where  $\boldsymbol{\xi}$  represents the input of the approximator. This can represent the state when approximating the value function, or it can represent the state and action, so that the approximator represents the Q-function. The  $\boldsymbol{w}$  are the parameters of the network.

#### Training the Critic

Different methods are possible to train the critic. They are all based on the temporal difference in (2.12) or (2.16).

- **TD( $\lambda$ )**

The TD( $\lambda$ ) update was introduced in section 2.2.4 as an extension to the update (2.12). For a function approximator the update is based on the gradient with respect to the parameters of the function approximator. This leads to an update like [68]:<sup>6</sup>

$$\boldsymbol{w}' = \boldsymbol{w} + \alpha \sum_{k=0}^{N-1} \Delta \boldsymbol{w}_k, \quad (2.22)$$

with

$$\Delta \boldsymbol{w}_k = (r_k + \gamma V(\boldsymbol{\xi}_{k+1}, \boldsymbol{w}) - V(\boldsymbol{\xi}_k, \boldsymbol{w})) \sum_{i=1}^k \lambda^{k-i} \nabla_{\boldsymbol{w}} V(\boldsymbol{\xi}_i, \boldsymbol{w}). \quad (2.23)$$

The  $\alpha$  is the learning rate and the  $\lambda$  is a weighting factor for past updates. The sequence of past updates indicate the eligibility trace and is used to make the update depend on the trajectory. This can speed up the training of the network. The value of  $0 < \lambda < 1$  for which it trains fastest is not known and depends on the problem. It can only be found empirically by repeating the training for different  $\lambda$ .

- **Minimize the quadratic temporal difference error**

Most learning methods for function approximators are based on minimizing the summed squared error between the target value and the network output. In a similar way the temporal difference can be used to express an error that is minimized by

---

<sup>6</sup>Note that the term “temporal difference” was introduced in [68], together with the TD( $\lambda$ ) learning rule for function approximators and not for the discrete case as described in section 2.2.4.

standard steepest decent methods [83]. This means the training of the critic becomes minimizing the quadratic temporal difference error.

$$E = \frac{1}{2} \sum_{k=0}^{N-1} (r_k + \gamma V(\boldsymbol{\xi}_{k+1}, \mathbf{w}) - V(\boldsymbol{\xi}_k, \mathbf{w}))^2 \quad (2.24)$$

where  $V$  represents the critic with weights  $\mathbf{w}$  and  $\gamma$  is the discount factor. We see here that this error does not completely specify the function to approximate. It gives the difference in output for two points in the input.

The steepest decent update rule can now be based on this error:

$$\mathbf{w}' = \mathbf{w} - \alpha \nabla_{\mathbf{w}} E = \mathbf{w} + \sum_{k=0}^{N-1} \Delta \mathbf{w}_k, \quad (2.25)$$

with

$$\Delta \mathbf{w}_k = (r_k + \gamma V(\boldsymbol{\xi}_{k+1}, \mathbf{w}) - V(\boldsymbol{\xi}_k, \mathbf{w})) (\gamma \nabla_{\mathbf{w}} V(\boldsymbol{\xi}_{k+1}, \mathbf{w}) - \nabla_{\mathbf{w}} V(\boldsymbol{\xi}_k, \mathbf{w})). \quad (2.26)$$

If the learning rate is small enough convergence to a suboptimal solution can be guaranteed. That this solution can not be guaranteed to be optimal is due to the possibility that it converges to a local minimum of (2.24).

- **Residual Algorithms**

The temporal difference method described above can be regarded as the residue of the Bellman equation, because if the Bellman equation would hold this error would be zero. The advantage of using steepest descent on the Bellman residual is that it can always be made to converge. This is just a matter of making the learning rate small enough. Of course there remains the risk of ending in a local minimum.

On the other hand, the learning converges very slowly compared to the TD( $\lambda$ ) update rule. This, however, has the disadvantage that the convergence for a function approximator cannot be guaranteed. Based on this observation Residual Algorithms were proposed [6]. The main idea is to combine TD( $\lambda$ ) learning with the minimization of the Bellman residual.

## Training the Actor

The second function approximator is the actor, which represents the policy or feedback function  $\mathbf{u}_k = \mathbf{g}(\mathbf{x}_k, \mathbf{w}_a)$ . Here  $\mathbf{w}_a$  represents the weights of the actor. In the discrete case the policy that is greedy with respect to the value function is chosen. Here the critic approximates the value function and the parameters of the actor should be chosen in such way that it is greedy with respect to the critic.

Since the critic is represented by a function approximator, it can have any form. This means that the values of the actor cannot be derived directly from the critic. When the actor network is represented by a continuous differential function, a steepest decent approach can be used to adapt the weights. For this there are two possibilities, depending on whether the critic represents a Q-function or a value function:

- **Backpropagation with respect to the critic**

If the control action is an input of the critic, the gradient with respect to the action can be computed. This indicates how a change in action influences the output of the critic. The critic can be regarded as an error function that indicates how the outputs of the actor should change. So the update of the weights of the actor can be performed according to

$$\Delta \mathbf{w}_a = -\nabla_{\mathbf{w}_a} \mathbf{g}(\mathbf{x}_k, \mathbf{w}_a) \nabla_{\mathbf{u}} Q(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}). \quad (2.27)$$

Note that the input  $\mathbf{u}_k$  of this network is the output of the network representing the actor.<sup>7</sup>

- **Backpropagation based on temporal difference**

The critic can be used to determine the temporal difference. Then the actor can be updated such that the temporal difference becomes smaller:

$$\Delta \mathbf{w}_a = -\nabla_{\mathbf{w}_a} \mathbf{g}(\boldsymbol{\xi}_k, \mathbf{w}_a) (r_k + \gamma V(\boldsymbol{\xi}_{k+1}, \mathbf{w}) - V(\boldsymbol{\xi}_k, \mathbf{w})). \quad (2.28)$$

In this case the critic and actor are trained based on the same temporal difference.

The backpropagation based on the temporal difference was first used in [9]. Then it was improved in [85]. Recently these approaches have gained more interest as solution methods for problem domain that are non-Markov [41][7][72]. The actor then represents the probability distribution from which the actions are drawn. The result is therefore not a deterministic policy. A drawback of these approaches is that they learn very slowly.

The general idea of the actor critic approaches has been formalized in Heuristic Dynamic Programming (HDP) [83]. This describes a family of approaches that are based on backpropagation and the actor critic configuration. The HDP and action depended HDP can be regarded as temporal difference learning and Q-learning. An alternative approach is to approximate the gradient of the critic that is used to update the actor. This has been extended to a more general framework in which the value of the critic and the gradient are trained simultaneously [83][56]. The main drawback of these approaches is that they become so complicated that even the implementation of some of these algorithms is not trivial [56].

### 2.3.3 Summary

In this section we showed that there are two different ways to apply RL algorithms to obtain a feedback function for systems with a continuous state and action space. One solution approach is based on discretizing the state space so that the original RL algorithms can be applied. The other solution is that function approximators are used to estimate the value or Q-function. In this case two approximators are required, one to represent the feedback and one to represent the value or Q-function.

<sup>7</sup>Therefore we could not express this with a single  $\boldsymbol{\xi}$  as input.

## 2.4 Discussion

Now we have described RL and control so we can refine our problem statement. First look at what RL and control have in common:

- RL is used to solve an optimal control task. This means that an optimal control task can be formulated into a RL problem.
- The system identification requires that the system is excited in order to estimate the parameters of the system. The RL algorithms require exploration in order to estimate the parameters of the value function.

In a sense RL can be seen as adaptive control where the parameters are estimated and used to tune the parameters of the controller.

There are some differences:

- Stability plays an important role in control theory, while in RL the set of feasible actions is assumed to be known in advanced.
- The state transitions in the model are the actual values of the state, while in the MDP framework the state transitions are given by the probabilities. This has influence on how the parameters are estimated.

Since we are interested in controlling systems with continuous state and action spaces, we have to choose one of the two main approaches. The state space quantization results in a feedback function that is not well defined. It is based on many local estimations. For the resulting feedback function is therefore not easy to see what the consequences are when this feedback is used to control a continuous time system.

The function approximator approaches are better suited for understanding the resulting feedback. This is because the resulting feedback function is already determined by the actor. The main problem with these approaches is that they rely on training two different approximators, so that it is hard to see how well they are trained. Also it is hard to see how the result can be improved.

The only function approximator approach that comes close to our requirements presented in section 1.3, is the approach that is applied to the linear quadratic regularization task. The only problem is that it does not apply to nonlinear systems. Still this approach will be a good starting point for our investigation.



# Chapter 3

## LQR using Q-Learning

### 3.1 Introduction

In this chapter we will present a theoretic framework that enables us to analyze the use of RL in problem domains with continuous state and action spaces. The first theoretical analysis and proof of convergence of RL applied to such problems can be found in [82]. It shows the convergence to the correct value function for a Linear Quadratic Regularization task, where the weights of a carefully chosen function approximator were adjusted to minimize the temporal difference error. Based on the same idea the convergence was proven for other RL approaches, including Q-learning [44].

In [16][19][20][17], a policy iteration based Q-learning approach was introduced to solve a LQR task. This was based on a Recursive Least Squares (RLS) estimation of a quadratic Q-function. These approaches do not use the model of the linear system and can be applied if the system is unknown. If data is generated with sufficient exploration, the convergence to the optimal linear feedback can be proven.

Both [17] and [44] indicate that the practical applicability of the results are limited by the absence of noise in the analysis. A scalar example in [17] shows that the noise introduces a bias in the estimation, making that proofs of convergence no longer hold. We are interested in the use of RL on real practical problem domains with continuous state and action spaces. This means that we have to include the noise in our analysis.

We are also interested in how well the RL approach performs compared to alternative solution methods. We can solve the LQR task with an unknown system using an indirect approach, where data is used to estimate the parameters of the system. Then these estimated parameters are used to compute the optimal feedback. Because we want to compare the results it is important to replace the RLS by a batch linear least squares estimation. This has the advantage that no initial parameters have to be specified and so the resulting solution only depends on the data and the solution method. The result is that both solution methods are off-line optimization methods, because first all data is generated and then the new feedback is computed.

According to the convergence proofs, sufficient exploration is required to find the op-

timal solution. This means that random actions have to be applied to the system. In a practical control task this is not desirable, so we need an indication of the minimal amount of exploration that is sufficient. This means that our analysis has to show how the performance of the two solution methods depend on the amount of exploration used to generate the data.

In the next section we will specify the Linear Quadratic Regularization task where the linear system is assumed to be unknown. We then present the two solution methods and give an overview on how to compare the performance of these two methods. Section 3.3 will focus on the influence of the exploration on the comparison. We will show that the noise determines the amount of exploration required for a guaranteed convergence. Also we will show that this amount of exploration differs for the two solution methods. The experimental confirmation of the results will be given in section 3.4, followed by the discussion and conclusion in section 3.5 and 3.6.

## 3.2 LQR with an Unknown System

In this section we will describe the LQR task and show how to obtain the optimal feedback when everything is known. We will then present a direct and an indirect solution method for the situation where the parameters of the system are unknown. Also we will define a performance measure and give an overview of the comparison of the two solution methods.

### 3.2.1 Linear Quadratic Regulation

In the Linear Quadratic Regulation (LQR) framework, the system is linear and the direct cost is quadratic. Let a linear time invariant discrete time system be given by:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{v}_k, \quad (3.1)$$

with  $\mathbf{x}_k \in \mathbb{R}^{n_x}$  the state,  $\mathbf{u}_k \in \mathbb{R}^{n_u}$  the control action and  $\mathbf{v}_k \in \mathbb{R}^{n_x}$  the system noise at time step  $k$ . All elements of system noise  $\mathbf{v}$  are assumed to be  $\mathcal{N}(0, \sigma_v^2)$  distributed and white. Matrix  $A \in \mathbb{R}^{n_x \times n_x}$  and  $B \in \mathbb{R}^{n_x \times n_u}$  are the parameters of the system.

The direct cost  $r$  is a quadratic function of the state and the control action at time  $k$ :

$$r_k = \mathbf{x}_k^T S \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k, \quad (3.2)$$

where  $S \in \mathbb{R}^{n_x \times n_x}$  and  $R \in \mathbb{R}^{n_u \times n_u}$  are the design choices. The objective is to find the mapping from state to control action ( $\mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$ ) that minimizes the total costs  $J$ , which is given by:

$$J = \sum_{k=0}^{\infty} r_k. \quad (3.3)$$

The value of  $J$  is finite if (3.2) approaches zero fast enough. This is the case when (3.1) is controlled using (1.3) and the closed loop is stable. The total costs  $J$  becomes infinite if the closed loop is unstable. It is possible to include a discount factor  $\gamma < 1$  in (3.3),

but then the total costs can be finite for an unstable system. We will use (3.3) without a discount factor (or  $\gamma = 1$ ), so that a finite  $J$  always implies that the system is stable.

The optimal control action  $\mathbf{u}^*$  is a linear function of the state:

$$\mathbf{u}_k^* = L^* \mathbf{x}_k \quad \text{with} \quad L^* = -(B^T K^* B + R)^{-1} B^T K^* A \quad (3.4)$$

where  $K^* \in \mathbb{R}^{n_x \times n_x}$  is the unique symmetric positive definite solution to the *Discrete Algebraic Riccati Equation* (DARE):

$$K^* = A^T (K^* - K^* B (B^T K^* B + R)^{-1} B^T K^*) A + S. \quad (3.5)$$

This solution exists if:  $(A, B)$  is controllable,  $(A, S^{\frac{1}{2}})$  is observable,  $S \geq 0$  (positive semi-definite) and  $R > 0$  (positive definite) [11]. Only with perfect knowledge about  $A$ ,  $B$ ,  $S$  and  $R$  can equation (3.5) be solved. This restricts the practical applicability of LQR because in practice perfect knowledge about  $A$ ,  $B$  is not available.

### 3.2.2 System Identification

In indirect adaptive control the parameters of the system have to be estimated, and we will refer to this as the System Identification (SI) approach.<sup>1</sup> This is our first method to solve the LQR problem with unknown  $A$  and  $B$ . The estimations are based on measurements generated by controlling the system using:

$$\mathbf{u}_k = L \mathbf{x}_k + \mathbf{e}_k \quad (3.6)$$

where  $L$  is the existing feedback<sup>2</sup> and  $\mathbf{e}_k \in \mathbb{R}^{n_u}$  represents the excitation (or exploration) noise. The main difference between  $\mathbf{e}$  and  $\mathbf{v}$  is that  $\mathbf{v}$  is an *unknown* property of the system, while  $\mathbf{e}$  is a random process that is *added on purpose* to the control action. All elements of  $\mathbf{e}$  are chosen to be  $\mathcal{N}(0, \sigma_e^2)$  distributed and white. Although the value of  $\mathbf{e}$  is always known, the two methods presented in this chapter will not use this knowledge.

Controlling the system for  $N$  time steps results in a set  $\{\mathbf{x}_k\}_{k=0}^N$  with:

$$\mathbf{x}_k = D_0 + \sum_{i=0}^{k-1} D^{k-i-1} (B \mathbf{e}_i + \mathbf{v}_i) \quad (3.7)$$

where  $D = A + BL$  represents the closed loop. The sets  $\{\mathbf{x}_k\}_{k=0}^N$  and  $\{\mathbf{u}_k\}_{k=0}^N$  (computed with (3.6)) form a data set that depends on the parameters of the system, the feedback, the initial state and both noise sequences. To estimate the parameters of the system, rewrite (3.1) to:

$$\mathbf{x}_{k+1}^T = \begin{bmatrix} \mathbf{x}_k^T & \mathbf{u}_k^T \end{bmatrix} \begin{bmatrix} A^T \\ B^T \end{bmatrix} + \mathbf{v}_k \quad (3.8)$$

<sup>1</sup>Actually closed-loop identification or identification for control would be more appropriate, but system identification stresses the main difference with the reinforcement learning method more.

<sup>2</sup>Note that this controller already has the form of (3.4), only the value of  $L$  is not optimal.

So for the total data set

$$Y_{\text{SI}} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0^T & \mathbf{u}_0^T \\ \vdots & \vdots \\ \mathbf{x}_{N-1}^T & \mathbf{u}_{N-1}^T \end{bmatrix} \begin{bmatrix} A^T \\ B^T \end{bmatrix} + \begin{bmatrix} \mathbf{v}_0^T \\ \vdots \\ \mathbf{v}_{N-1}^T \end{bmatrix} = X_{\text{SI}}\theta_{\text{SI}} + V_{\text{SI}} \quad (3.9)$$

should hold. Since  $V_{\text{SI}}$  is not known, only a least squares estimate of  $\theta_{\text{SI}}$  can be given:

$$\hat{\theta}_{\text{SI}} = (X_{\text{SI}}^T X_{\text{SI}})^{-1} X_{\text{SI}}^T Y_{\text{SI}}. \quad (3.10)$$

The estimated parameters of the system,  $\hat{A}$  and  $\hat{B}$ , can be derived from  $\hat{\theta}_{\text{SI}}$ . When  $\hat{A}$  and  $\hat{B}$  are used, the solution of (3.5) will be  $\hat{K}$ . Then a feedback  $\hat{L}_{\text{SI}}$  can be computed using (3.4). This feedback  $\hat{L}_{\text{SI}}$  is the resulting approximation of  $L^*$  by the SI approach.

### 3.2.3 The Q-function

Reinforcement learning is our second method for solving the LQR task with unknown  $A$  and  $B$ . This means that the costs in (3.2) will be regarded as the reinforcements. As explained in chapter 2, the main idea behind RL is to approximate the future costs and find a feedback that minimizes these costs. In LQR the solution of the DARE (3.5) can be used to express the future costs as a function of the state when the optimal feedback is used [11]:

$$V^*(\mathbf{x}_k) = \sum_{i=k}^{\infty} r_i = \mathbf{x}_k^T K^* \mathbf{x}_k, \quad (3.11)$$

with  $V^* : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ . The feedback that minimizes (3.11) is given by (3.4), which requires knowledge about  $A$  and  $B$ . So, it is not very useful<sup>3</sup> to estimate the parameters  $K^*$  of (3.11). Q-Learning is more appropriate, since it does not require knowledge about the system to obtain the feedback.

In Q-Learning the feedback is derived from the Q-function, which represents the future costs as a function of the state *and* action. So  $Q : \mathbb{R}^{n_x \times n_u} \rightarrow \mathbb{R}$  is the function to approximate based on the measurements. If we know what function we have to approximate, then we only have to estimate the parameters. According to (2.14)

$$V^*(\mathbf{x}) = \min_{\mathbf{u}} Q^*(\mathbf{x}, \mathbf{u}) = Q^*(\mathbf{x}, \mathbf{u}^*) \quad (3.12)$$

should hold. It can be shown [19][44] that  $Q^*(\mathbf{x}_k, \mathbf{u}_k^*)$  is given by:

$$Q^*(\mathbf{x}_k, \mathbf{u}_k^*) = \sum_{i=k}^{\infty} r_i = \begin{bmatrix} \mathbf{x}_k^T & \mathbf{u}_k^{*\top} \end{bmatrix} \begin{bmatrix} S + A^T K^* A & A^T K^* B \\ B^T K^* A & R + B^T K^* B \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k^* \end{bmatrix} \quad (3.13)$$

$$= \begin{bmatrix} \mathbf{x}_k^T & \mathbf{u}_k^{*\top} \end{bmatrix} \begin{bmatrix} H_{xx}^* & H_{xu}^* \\ H_{ux}^* & H_{uu}^* \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k^* \end{bmatrix} = \phi_k^{*\top} H^* \phi_k^*. \quad (3.14)$$

<sup>3</sup>In optimization tasks other than LQR, the computation of the future costs may be intractable so that an approximation of the future costs may be useful.

The vector  $\phi_k^{*\text{T}} = [\mathbf{x}_k^{\text{T}} \quad \mathbf{u}_k^{*\text{T}}]$  is the concatenation of the state and optimal control action and the matrix  $H^*$  contains the parameters of the optimal Q-function. This shows that the optimal Q-function for the LQR task is a quadratic function of the state and action.

The Q-function in (3.14) can be used to compute the optimal control action without the use of the system model. According to (2.15),

$$\mathbf{u}_k^* = \arg \min_{\mathbf{u}} Q^*(\mathbf{x}_k, \mathbf{u}) \quad (3.15)$$

should be computed for all states to get the optimal feedback function. The Q-function in (3.14) is a quadratic function and  $H^*$  is a symmetric positive definite matrix. So this function can easily be minimized by setting the derivative to the control action to zero:  $\nabla_{\mathbf{u}_k^*} Q^*(\mathbf{x}_k, \mathbf{u}_k^*) = 2H_{ux}^* \mathbf{x}_k + 2H_{uu}^* \mathbf{u}_k^* = 0$ , resulting in:

$$\mathbf{u}_k^* = -(H_{uu}^*)^{-1} H_{ux}^* \mathbf{x}_k = L^* \mathbf{x}_k \quad \text{with } L^* = -(H_{uu}^*)^{-1} H_{ux}^*. \quad (3.16)$$

With the  $H_{ux}^* = B^{\text{T}} K^* A$  and  $H_{uu}^* = R + B^{\text{T}} K^* B$  in (3.14), this result is identical to (3.4).

It is not the optimal Q-function that is being approximated, but the function representing the future costs. This is the function  $Q^L(\mathbf{x}, \mathbf{u}) = \phi^{\text{T}} H^L \phi$  (with  $\phi^{\text{T}} = [\mathbf{x} \quad \mathbf{u}]^{\text{T}}$ )<sup>4</sup>, because all measurements are generated using some feedback  $L$ . The  $H^L$  is symmetric and positive definite so that  $L' = -(H_{uu}^L)^{-1} H_{ux}^L$ <sup>5</sup> is the feedback that minimizes  $Q^L$ . The  $L'$  does not have to be the optimal feedback but it will have lower future costs than  $L$ . Estimating the parameters of  $Q^L$  forms a policy evaluation step and computing  $L'$  forms a policy improvement step. This means that this Q-Learning approach is based on policy iteration. If  $L'$  is not good enough, the whole procedure can be repeated by generating measurements using  $L'$ . If the parameters of  $Q^L$  are always estimated correctly, the sequence of new values of  $L'$  forms a contraction towards the optimal solution  $L^*$  [44]. This means we only have to verify the correctness of one policy improvement step. Then the convergence to the optimal solution follows from induction.

### 3.2.4 Q-Learning

In (2.16) the update rule for Q-Learning is given. It is based on repeatedly restarting the system and generating new data in each run. The update also has a learning rate that has to decrease according to (2.19) and (2.20). This makes it impossible to compare the result with that of the SI approach. We therefore have to change the Q-learning algorithm such that it uses one single data set and does not use a learning rate.

The parameters  $H^L$  of the Q-function should be estimated in the same way as the parameters of the system in paragraph 3.2.2. So the same data set with  $\{\mathbf{x}_k\}_{k=0}^N$  and  $\{\mathbf{u}_k\}_{k=0}^N$  is used. Q-Learning also uses scalar reinforcements. These are the direct costs

<sup>4</sup>If there is also noise:  $Q^L(\mathbf{x}, \mathbf{u}) = \phi^{\text{T}} H^L \phi + \mathbf{v}^{\text{T}} K^L \mathbf{v}$

<sup>5</sup>The  $'$  indicates the feedback that is optimal according to the Q-function, so  $L'$  optimizes  $Q^L$ . This is equivalent to the *greedy policy* described in chapter 2.

computed with (3.2).<sup>6</sup> The parameters of  $Q^L$  are estimated based on the data set, generated using feedback  $L$ . The function  $Q^L$  can be estimated by writing its definition recursively:

$$Q^L(\mathbf{x}_k, \mathbf{u}_k) = \sum_{i=k}^{\infty} r_i = r_k + \sum_{i=k+1}^{\infty} r_i = r_k + Q^L(\mathbf{x}_{k+1}, L\mathbf{x}_{k+1}). \quad (3.17)$$

Note that this definition implies that the data is generated using a stabilizing feedback. In case the feedback  $L$  is not stable, the function  $Q^L(\mathbf{x}_k, \mathbf{u}_k)$  is not defined because the sum of future reinforcement is not bounded. Therefore the correct values of  $H^L$  do not exist and cannot be estimated.

From (3.17) it follows that:

$$r_k + Q^L(\mathbf{x}_{k+1}, L\mathbf{x}_{k+1}) - Q^L(\mathbf{x}_k, \mathbf{u}_k) = 0. \quad (3.18)$$

If in this equation  $Q^L$  is replaced by its approximation  $\hat{Q}^L$  the left hand side is the Temporal Difference (TD). Because both functions are quadratic, the right hand side of (3.18) is only zero if  $\hat{Q}^L$  has the same parameters as  $Q^L$ . The parameters of  $\hat{Q}^L$  can be estimated by reducing the distance between the TD and zero. This can be formulated as a least squares estimation as in (3.10). We define:

$$\phi_k^T = [\mathbf{x}_k^T \quad \mathbf{u}_k^T], \quad \phi_{k+1}^T = [\mathbf{x}_k^T \quad L^T \mathbf{x}_k^T] \quad \text{and} \quad \mathbf{w}_k^T = \mathbf{v}_k^T K^L \mathbf{v}_k - \mathbf{v}_{k+1}^T K^L \mathbf{v}_{k+1}. \quad (3.19)$$

Note that the definition of  $\phi_{k+1}^T$  is slightly different from  $\phi_k^T$ . It is possible to write (3.18) as:<sup>7</sup>

$$r_k = Q^L(\mathbf{x}_k, \mathbf{u}_k) - Q^L(\mathbf{x}_{k+1}, L\mathbf{x}_{k+1}) \quad (3.20)$$

$$= \phi_k^T H^L \phi_k - \phi_{k+1}^T H^L \phi_{k+1} + \mathbf{w}_k \quad (3.21)$$

$$= \text{vec}^\triangleleft(\phi_k \phi_k^T)^T \text{vec}^\triangleleft(H^L) - \text{vec}^\triangleleft(\phi_{k+1} \phi_{k+1}^T)^T \text{vec}^\triangleleft(H^L) + \mathbf{w}_k \quad (3.22)$$

$$= \text{vec}^\triangleleft(\phi_k \phi_k^T - \phi_{k+1} \phi_{k+1}^T)^T \text{vec}^\triangleleft(H^L) + \mathbf{w}_k = \text{vec}^\triangleleft(\Phi_k) \text{vec}^\triangleleft(H^L) + \mathbf{w}_k. \quad (3.23)$$

Note that the matrix  $\Phi_k$  also depends on  $L$ . For all time steps the following holds:

$$Y_{\text{QL}} = \begin{bmatrix} r_0 \\ \vdots \\ r_{N-1} \end{bmatrix} = \begin{bmatrix} \text{vec}^\triangleleft(\Phi_0)^T \\ \vdots \\ \text{vec}^\triangleleft(\Phi_{N-1})^T \end{bmatrix} \text{vec}^\triangleleft(H^L) + \begin{bmatrix} \mathbf{w}_0 \\ \vdots \\ \mathbf{w}_{N-1} \end{bmatrix} = X_{\text{QL}} \theta_{\text{QL}} + V_{\text{QL}}, \quad (3.24)$$

so that

$$\hat{\theta}_{\text{QL}} = (X_{\text{QL}}^T X_{\text{QL}})^{-1} X_{\text{QL}}^T Y_{\text{QL}} \quad (3.25)$$

<sup>6</sup>Note that for the SI approach the parameters of the system are unknown, but still the weighting of the design matrices can be made. Conceptually this does not make any sense. In a practical situation it is more likely that some scalar indication of performance is available, like for instance the energy consumption of the system. We compute the direct cost using (3.2) for fair comparison.

<sup>7</sup>Define  $\text{vec}^\triangleleft(A)$  as the function that stacks the upper triangle elements of matrix  $A$  into a vector.

gives an estimation of  $\text{vec}^\triangleleft(H^L)$ . Since  $\hat{H}$  should be symmetrical it can be derived from  $\text{vec}^\triangleleft(H^L)$ . By applying (3.16) to matrix  $\hat{H}$  the resulting feedback  $\hat{L}_{\text{QL}}$  for the Q-Learning approach is computed. This should be an approximation of  $L'$ .

This variant of Q-learning only applies to the LQR framework, therefore we will refer to it as the Linear Quadratic Regulation Q-Learning (LQRQL) approach.<sup>8</sup> The main difference is that it does not require restarting the system several times to generate sufficient data for the correct estimation of all Q-values. Instead it uses prior knowledge about the function class of the Q-function that is chosen to fit the LQR task. This can be seen as generalization over the state-action space, which for the LQR task is globally correct. This also holds for the approaches in [17][44]. The only difference between these approaches and our approach using (3.25), is that we choose to learn in one step using the entire data set. The main reason for doing this is to make it possible to compare the result with the SI approach. Also the analysis of the outcome is easier when the estimation is not performed recursively.

### 3.2.5 The Performance Measure

We have described two different methods that use measurements to optimize a feedback resulting in  $\hat{L}_{\text{SI}}$  and  $\hat{L}_{\text{QL}}$ . For the comparison a scalar performance measure is required to indicate which of these feedbacks performs best. There are three ways to measure the performance:

- **Experimental:** Run the system with the resulting feedbacks and compute the total costs. The performances of both approaches can only be compared for one specific setup and it does not indicate how “optimal” the result is. For a real system, this is the only possible way to compare the results.
- **Optimal Feedback:** In a simulation there is knowledge of  $A$  and  $B$ , so the optimal feedback  $L^*$  can be computed using (3.4). A norm<sup>9</sup>  $\|L^* - \hat{L}\|$  will not be a good performance measure because feedbacks with similar  $\|L^* - \hat{L}\|$  can have different future costs. It is even possible that if  $\|L^* - \hat{L}_1\| < \|L^* - \hat{L}_2\|$ ,  $\hat{L}_1$  results in an unstable closed loop while  $\hat{L}_2$  results in a stable closed loop. This means that this measure can be used to show that the resulting feedback approaches the optimal feedback, but this does not show that this will result in lower total costs.
- **DARE Solution:** Knowledge about  $A$  and  $B$  can be used in (3.5) to compute the solution of the DARE  $K^*$ . This gives the future costs (3.11) when starting in  $\mathbf{x}_0$  and using  $L^*$ . The costs when using an approximated feedback  $\hat{L}$  can be expressed also like (3.11), but then with matrix  $K^{\hat{L}}$ . Comparing the matrices  $K^*$  and  $K^{\hat{L}}$  results in a performance indication that only depends on the initial state.

<sup>8</sup>This should not be confused with the Least Squares TD approach [18][13], that applies to MDPs.

<sup>9</sup>Here  $\hat{L}$  indicates the resulting feedback of both approaches.

We will define a performance measure based on the DARE solution, because it is the least sensitive to the settings of the experiment.

When using  $\hat{L}$ , the value function  $V^{\hat{L}}(\mathbf{x}_0)$  gives the total costs  $J^{\hat{L}}$  when starting in state  $\mathbf{x}_0$ . It is given by:

$$V^{\hat{L}}(\mathbf{x}_0) = \sum_{k=0}^{\infty} r_k = \mathbf{x}_0^T \sum_{k=0}^{\infty} (A^T + \hat{L}^T B^T)^k (S + \hat{L}^T R \hat{L}) (A + B \hat{L})^k \mathbf{x}_0 = \mathbf{x}_0^T K^{\hat{L}} \mathbf{x}_0 \quad (3.26)$$

where  $K^{\hat{L}}$  is again a symmetric matrix. (It is clear that this matrix only exists when the closed loop  $A + B \hat{L}$  has all its eigenvalues in the unit disc). When  $L^*$  is used the total costs  $V^*(\mathbf{x}_0)$  can be computed using (3.11). Let the relative performance (RP)  $\rho(\mathbf{x}_0)$  be the quotient between  $V^{\hat{L}}(\mathbf{x}_0)$  and  $V^*(\mathbf{x}_0)$ , so:

$$\rho^{\hat{L}}(\mathbf{x}_0) = \frac{V^{\hat{L}}(\mathbf{x}_0)}{V^*(\mathbf{x}_0)} = \frac{\mathbf{x}_0^T K^{\hat{L}} \mathbf{x}_0}{\mathbf{x}_0^T K^* \mathbf{x}_0} = \frac{\mathbf{x}_0^T (K^*)^{-1} K^{\hat{L}} \mathbf{x}_0}{\mathbf{x}_0^T \mathbf{x}_0} = \frac{\mathbf{x}_0^T \Gamma^{\hat{L}} \mathbf{x}_0}{\mathbf{x}_0^T \mathbf{x}_0}, \quad (3.27)$$

where  $\Gamma^{\hat{L}} = (K^*)^{-1} K^{\hat{L}}$ . Only when  $\hat{L} = L^*$ ,  $\Gamma^{\hat{L}}$  is the unit matrix. The RP  $\rho^{\hat{L}}(\mathbf{x}_0)$  is bounded below and above by the minimal and maximal eigenvalues of  $\Gamma^{\hat{L}}$ :

$$\rho_{\min}^{\hat{L}} = \lambda_{\min}(\Gamma^{\hat{L}}) \leq \rho^{\hat{L}}(\mathbf{x}_0) \leq \lambda_{\max}(\Gamma^{\hat{L}}) = \rho_{\max}^{\hat{L}} \quad \forall \mathbf{x}_0 \neq \mathbf{0} \quad (3.28)$$

Note that  $\rho_{\min}^{\hat{L}} = \rho_{\max}^{\hat{L}} = 1$  if and only if  $\hat{L} = L^*$  and that  $\rho_{\min}^{\hat{L}} \geq 1 \forall \hat{L}$ .

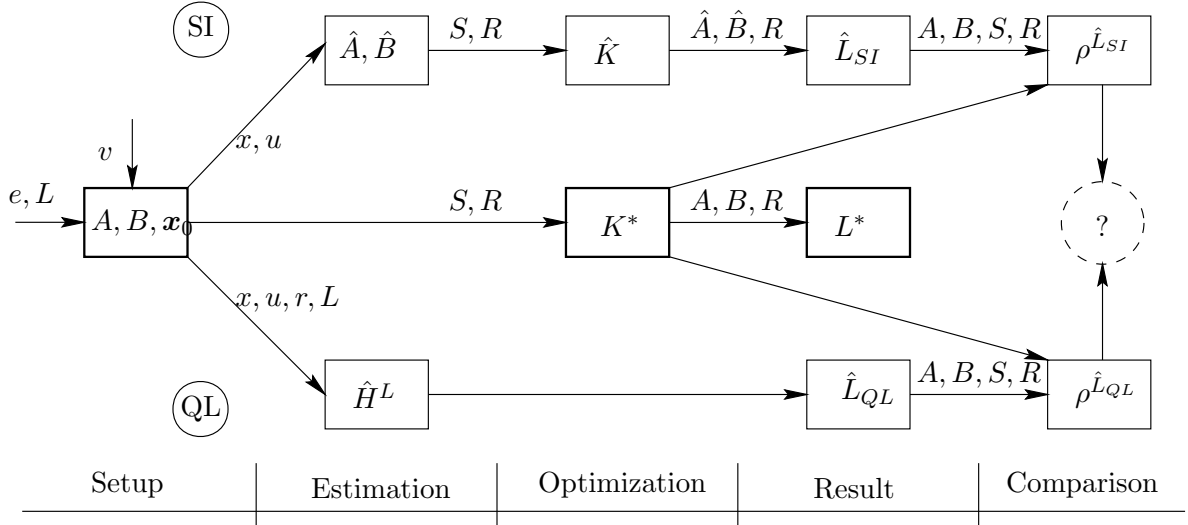
According to (3.28) three possible measures for the RP can be used:  $\rho_{\min}^{\hat{L}}$ ,  $\rho_{\max}^{\hat{L}}$  or  $\rho^{\hat{L}}(\mathbf{x}_0)$ . It does not matter for the comparison which measure is used, so in general we will use  $\rho^{\hat{L}}$  to indicate one of these measures. Note that  $\rho_{\min}^{\hat{L}}$  and  $\rho_{\max}^{\hat{L}}$  only depend on the feedback  $\hat{L}$  and the four matrices  $A$ ,  $B$ ,  $S$  and  $R$  that define the problem. In a practical situation  $\rho_{\max}^{\hat{L}}$  seems the best choice because it represents the worst case RP with respect to  $\mathbf{x}_0$ . In this chapter we will call feedback  $L_1$  better than feedback  $L_2$  if  $\rho^{L_1} < \rho^{L_2}$ .

### 3.2.6 Overview

The schematic overview in figure 3.1 summarizes this section. The setup at the left shows the system parameters and noise, but also the feedback  $L$  and exploration noise  $\mathbf{e}$  to generate the measurements indicated with  $\mathbf{x}$ ,  $\mathbf{u}$  and  $r$  (note that  $r$  is computed using  $S$  and  $R$ ). The SI approach is shown at the top and the Q-Learning approach at the bottom. The computation of the optimal feedback using  $A$  and  $B$  is shown in the middle.

For LQRQL figure 3.1 shows no explicit optimization, because this is implicitly included in the estimation of the Q-function. This is the difference between the SI and the QL approach: the SI approach is a two step method, where the estimation and the optimization are performed independently. LQRQL is a one step method, where estimation and optimization are performed at once. Figure 3.1 also shows that no additional information is required to derive  $\hat{L}_{\text{QL}}$  from  $H^L$ .

The question mark at the very right of figure 3.1 indicates the comparison between  $\rho^{\hat{L}_{\text{SI}}}$  and  $\rho^{\hat{L}_{\text{QL}}}$ . In the next section we will relate this comparison to the amount of exploration, indicated with the  $e$  at the very left of figure 3.1.



**Figure 3.1. The Overview.** The boxes indicate the parameters and the symbols next to the arrows indicate the required information to compute the next result.

### 3.3 The Influence of Exploration

In this section we will investigate the influence of the exploration on the relative performances and the comparison. We will start in 3.3.1 by reformulating the method of the estimation to make it possible to express the estimation error of the linear least squares estimation. In 3.3.2 and 3.3.3 the influence of the exploration is investigated for both methods. In 3.3.4 the exploration characteristic will be introduced to describe the influence of the exploration on the performance of the resulting feedbacks.

#### 3.3.1 The estimation reformulated

Both approaches described in the previous section, are based on a linear least squares estimation. The equations (3.10) and (3.25) can be written as:

$$\hat{\theta} = (X^T X)^{-1} X^T Y. \quad (3.29)$$

This solution  $\hat{\theta}$  depends only on the matrices  $X$  and  $Y$ , so no additional parameters influence the result. For a fair comparison this is an important property. In practice (3.29) is hardly ever used because of its poor numerical performance. Different decomposition methods exist to overcome numerical problems. This is important for the implementation of the simulations, but we will also use it to investigate the influence of the exploration. The matrix inversion in (3.29) is the main problem for our analysis, because this makes it very hard to see how the exploration influences the estimation.

In QR-decomposition<sup>10</sup>  $X$  is decomposed into an upper triangular square matrix  $M$  and a unitary matrix  $Z$ , so that (3.29) can be written as:

$$\hat{\theta} = ((ZM)^T ZM)^{-1} (ZM)^T Y = M^{-1} Z^T Y. \quad (3.30)$$

This is sufficient for an efficient implementation but it still uses a matrix inversion, making it hard to see how this solution depends on the exploration. To see the influence of the exploration, this solution has to be rearranged even more.

The definition of  $Z$  and  $M$  in appendix A.1 makes use of projection matrices  $P$ . Let  $P_i$  be the projection matrix corresponding to  $X_{*i}$ , which represents the  $i^{\text{th}}$  column of  $X$ , then  $P_i$  can be defined recursively according to:

$$P_i = P_{i-1} - \frac{P_{i-1} X_{*i-1} X_{*i-1}^T P_{i-1}^T}{\|P_{i-1} X_{*i-1}\|_2^2} \quad \text{and} \quad P_1 = I. \quad (3.31)$$

So  $P_i$  depends on all columns of  $X$  from  $X_{*1}$  to  $X_{*i-1}$ . Multiplying these columns with  $P_i$  results in a zero vector, so the part of  $X_{*i}$  that is a linear combination of the columns  $X_{*1}$  to  $X_{*i-1}$  does not contribute to outcome of  $P_i X_{*j}$ .

Appendix A.2 shows that matrices  $P$  can be used to solve (3.30) without matrix inversion. Let  $\hat{\theta}_{n*}$  be the last row and  $\hat{\theta}_{i*}$  be the  $i^{\text{th}}$  row, then they are given by:<sup>11</sup>

$$\hat{\theta}_{n*} = \frac{X_{*n}^T P_n^T}{\|P_n X_{*n}\|_2^2} Y \quad (3.32)$$

$$\hat{\theta}_{i*} = \frac{X_{*i}^T P_i^T}{\|P_i X_{*i}\|_2^2} \left( Y - \sum_{j=i+1}^n X_{*j} \hat{\theta}_{*j} \right) \quad \text{for } i < n. \quad (3.33)$$

So  $\hat{\theta}$  can be obtained recursively by starting at the last row. If one of the columns of  $X$  is a linear combination of all other columns then  $(X^T X)^{-1}$  is singular. In this situation  $P_i X_{*i}$  in (3.33) will become zero, resulting in a singularity as well.<sup>12</sup> We will use (3.33) only for the theoretical analysis of the exploration and not for the implementation, because its numerical performance is even worse than (3.29).

### 3.3.2 The System Identification approach

We first rewrite the estimation for the SI approach and show how the resulting feedback depends on the estimation. We then express the estimation error and show how it depends on the exploration. Finally we show the consequences of the estimation error on the resulting feedback and its performance.

<sup>10</sup>The name refers to matrices  $Q$  and  $R$ , but we will use  $Z$  and  $M$  because we already use the symbols  $Q$  and  $R$ .

<sup>11</sup>In the rest of the chapter we will ignore the absence of the sum term for the  $n^{\text{th}}$  row by defining a dummy  $\hat{\theta}_{n+1*}$  that equals zero.

<sup>12</sup>The scalar  $\|P_i X_{*i}\|_2^2$  in (3.33) is squared, so it will go faster to zero than the elements of vector  $X_{*i}^T P_i^T$ .

### The Estimation

To show the influence of the exploration, matrix  $X_{\text{SI}}$  in (3.9) can be split in a part that depends on  $\mathbf{x}$  and in a part that depends on  $\mathbf{u}$ :

$$X_{\text{SI}} = \begin{bmatrix} \mathcal{X} & \mathcal{U} \end{bmatrix} = \begin{bmatrix} \mathcal{X} & \mathcal{X}L^T + \mathcal{E} \end{bmatrix} \quad \text{with} \quad \mathcal{X} = \begin{bmatrix} \mathbf{x}_0^T \\ \vdots \\ \mathbf{x}_{N-1}^T \end{bmatrix} \quad \mathcal{U} = \begin{bmatrix} \mathbf{u}_0^T \\ \vdots \\ \mathbf{u}_{N-1}^T \end{bmatrix} \quad \mathcal{E} = \begin{bmatrix} \mathbf{e}_0^T \\ \vdots \\ \mathbf{e}_{N-1}^T \end{bmatrix}. \quad (3.34)$$

Also the control actions are split into a feedback part and an exploration part, but according to (3.9) some exploration is still contained in  $\mathcal{X}$  and  $Y_{\text{SI}}$ .

Appendix A.3 shows that the columns of  $\hat{B}$  and  $\hat{A}$  are given by:

$$\hat{B}_{*i} = \frac{\mathcal{E}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=n_x+i+1}^{n_x+n_u} \mathcal{U}_{*j} \hat{B}_{*j}) \quad (3.35)$$

$$\hat{A}_{*i} = \frac{\mathcal{X}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{A}_{*j} - \mathcal{U} \hat{B}^T). \quad (3.36)$$

Without exploration the value of  $\hat{B}$  becomes infinite, because  $\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2$  approaches zero faster than  $\mathcal{E}_{*i}^T P_{n_x+i}^T$ . This also makes  $\hat{A}$  infinite. For low exploration the term  $\mathcal{U} \hat{B}^T$  dominates the outcome of (3.36). Then  $\hat{A}$  becomes more linear dependent on  $\hat{B}$ . So for low exploration ( $\hat{A}, \hat{B}$ ) are more likely to be uncontrollable.

Appendix A.3 also shows that the columns of  $\hat{D} = \hat{A} + L\hat{B}$  are given by:

$$\hat{D}_{*i} = \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \mathcal{E} \hat{B}^T - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{D}_{*j}). \quad (3.37)$$

Because  $\hat{B}$  is multiplied with  $\mathcal{E}$ ,  $\hat{D}$  does not become very large for low amounts of exploration. Therefore we will use  $\hat{B}$  and  $\hat{D}$  to obtain the resulting feedback  $\hat{L}_{\text{SI}}$ .

### The Feedback

To compute the feedback (3.5) should be solved using  $\hat{A}$  and  $\hat{B}$ , so:

$$\hat{K} = \hat{A}^T (\hat{K} - \hat{K} \hat{B} (\hat{B}^T \hat{K} \hat{B} + R)^{-1} \hat{B}^T \hat{K}) \hat{A} + S. \quad (3.38)$$

A unique solution  $\hat{K}$  to (3.38) does not have to exist, especially when  $\hat{A}$  and  $\hat{B}$  are too large due to insufficient exploration. In this case the right hand side of (3.38) will become very small making  $\hat{K} \approx S$ . So we will assume that  $K^* - \hat{K}$  is not too large. The feedback is computed according to (3.4) using the estimated matrices:

$$\hat{L}_{\text{SI}} = -(R + \hat{B}^T \hat{K} \hat{B})^{-1} \hat{B}^T \hat{K} \hat{A} = (R + \hat{B}^T \hat{K} \hat{B})^{-1} \hat{B}^T \hat{K} (\hat{B}L - \hat{D}). \quad (3.39)$$

By replacing  $\hat{A}$  by  $\hat{B}L - \hat{D}$ , two possible outcomes can already be given:

- **Too low exploration:**  $\hat{L}_{\text{SI}} \approx L$

If the amount of exploration is much too low,  $\hat{B}$  in (3.39) becomes much too large because of the low value of  $\mathcal{E}$  in (3.35). So  $\hat{D}$  and  $R$  can be neglected, resulting in  $\hat{L}_{\text{SI}} \approx (\hat{B}^T \hat{K} \hat{B})^{-1} \hat{B}^T \hat{K} \hat{B} L = L$ . This means that the outcome will approximately be the feedback that was used to generate the data!

- **High exploration:**  $\hat{L}_{\text{SI}} \approx L^*$

For very high amounts of exploration the system noise  $V_{\text{SI}}$  in (3.9) can be neglected. The least squares estimation will almost be perfect, so solving the DARE and computing feedback will approximately have the optimal feedback  $L^*$  as outcome.

We can conclude that for insufficient exploration the relative performance does not change and for abundant exploration the relative performance approaches one. We will determine the minimal amount of exploration required to obtain the second outcome.

### The Estimation Error

By defining  $\mathbf{y}_k = \mathbf{x}_{k+1}$  and using (3.7), it is possible to write:

$$\mathbf{y}_k = D^{k+1} \mathbf{x}_0 + \sum_{i=0}^k D^{k-i} (B \mathbf{e}_i + \mathbf{v}_i). \quad (3.40)$$

So  $Y_{\text{SI}}$  can be written as:

$$Y_{\text{SI}} = \mathcal{X} D^T + \mathcal{E} B^T + V_{\text{SI}}. \quad (3.41)$$

This can be used to get an expression for the error in the estimations of  $B$  and  $D$ .

$$\hat{B}_{*i} = \frac{\mathcal{E}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2} (\mathcal{X} D^T + \mathcal{E} B^T + V_{\text{SI}} - \sum_{j=n_x+i+1}^{n_x+n_u} \mathcal{U}_{*j} \hat{B}_{*j}) \quad (3.42)$$

$$= \frac{\mathcal{E}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2} (\mathcal{E} B^T + V_{\text{SI}} - \sum_{j=n_x+i+1}^{n_x+n_u} \mathcal{E}_{*j} \hat{B}_{*j}) \quad (3.43)$$

$$= B_{*i} + \frac{\mathcal{E}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2} (V_{\text{SI}} - \sum_{j=n_x+i+1}^{n_x+n_u} \mathcal{E}_{*j} (\hat{B}_{*j} - B_{*j})). \quad (3.44)$$

So the estimation error  $\bar{B}_{*i} = \hat{B}_{*i} - B_{*i}$  is given by:

$$\bar{B}_{*i} = \frac{\mathcal{E}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2} (V_{\text{SI}} - \sum_{j=n_x+i+1}^{n_x+n_u} \mathcal{E}_{*j} \bar{B}_{*j}) \quad (3.45)$$

In the same way:

$$\hat{D}_{*i} = \frac{\mathcal{X}_i^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (\mathcal{X} D^T + \mathcal{E} B^T + V_{\text{SI}} - \mathcal{E} \hat{B}^T - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{D}_{*j}) \quad (3.46)$$

$$\bar{D}_{*i} = \frac{\mathcal{X}_i^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (V_{\text{SI}} - \mathcal{E} \bar{B}^T - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \bar{D}_{*j}). \quad (3.47)$$

The estimation errors depend on the exploration noise in  $\mathcal{E}$  and the system noise in  $V_{\text{SI}}$ .

The estimations  $\hat{B}$  and  $\hat{D}$  can be written as the sum of the correct value and the estimation error. So  $\hat{B} = B + \bar{B}$  and  $\hat{D} = D + \bar{D}$ . These expressions can be used in (3.39) to see how the resulting feedback depends on the exploration and system noise, because the correct values  $B$  and  $D$  do not depend on  $\mathcal{E}$  and  $V_{\text{SI}}$ .

### The Minimal Exploration

Expressions (3.45) and (3.47) hold for any  $\mathcal{E}$  and  $V_{\text{SI}}$ . To focus on the amounts of exploration and system noise, the estimation errors should be expressed using  $\sigma_e$  and  $\sigma_v$ . These errors depend on the configuration so only an indication of the level of magnitude of these errors can be given.

We have to make the following assumptions:

$$E\{\mathcal{E}_{*i}^T \mathcal{E}_{*i}\} \sim c_1 N \sigma_e^2 \quad (3.48)$$

$$E\{\mathcal{E}_{*i}^T V_{\text{SI}*i}\} \sim c_2 N \sigma_e \sigma_v \quad (3.49)$$

$$E\{\|P_i \mathcal{X}_{*i}\|_2^2\} \sim c_3 + c_4 \sigma_v^2 + c_5 \sigma_v \sigma_e + c_6 \sigma_e^2, \quad (3.50)$$

with  $c_1 \cdots c_6$  constants depending on  $n_x$ ,  $n_u$  and  $B$ . Note that these assumptions are rude approximations that indicate the expectation value over a time interval of size  $N$ . Assumption (3.49) indicates that the level of magnitude of the expectation value is proportional to the cross correlation between  $\mathcal{E}_{*i}$  and  $V_{\text{SI}*i}$ . Given the fixed time interval, this value may vary a lot depending on the particular noise sequences. This means that the constant  $c_2$  depends very much on these noise sequences. The same holds for  $c_5$  in (3.49). The constant  $c_3$  is included to incorporate the dependency on the initial state  $\mathbf{x}_0$ .

Now it is possible to give an approximation of the expected estimation errors (3.45) and (3.47). They are proportional to:

$$E\{\bar{B}\} \sim \frac{c_2 \sigma_v}{c_1 \sigma_e} \quad (3.51)$$

$$E\{\bar{D}\} \sim \frac{\sigma_v}{\sqrt{c_3 + c_4 \sigma_v^2 + c_5 \sigma_v \sigma_e + c_6 \sigma_e^2}}. \quad (3.52)$$

Note that  $E\{\bar{D}\}$  has a maximum for  $\sigma_e = -\frac{c_5 \sigma_v}{2c_6} \sim \sigma_v$ , which in general is slightly less than  $\sigma_v$ .

The errors in (3.52) are zero if  $\sigma_v = 0$ , so exploration is only required to prevent singularity in the computations of the least squares estimate.

For  $\sigma_v \neq 0$  it is possible to neglect  $\hat{D}$  and  $R$  in (3.39) if  $\bar{B}$  makes  $\hat{B}$  much too large. The maximum of  $E\{\bar{D}\}$  is less than one and  $E\{\bar{B}\}$  is also less than one, so for  $\sigma_e \approx \sigma_v$  the  $\hat{D}$  and  $R$  in (3.39) cannot be neglected. This is the minimal amount of exploration that is required, for more exploration the estimations will almost be correct. So as a rule of thumb:

*The amount of exploration should be larger than the amount of system noise!*

### 3.3.3 The LQRQL approach

Analog to the SI approach, we will determine the dependency of the estimation errors on the exploration and system noise. We also will show the consequences of the estimation error on the resulting feedback and its performance.

#### The Estimation

To show the influence of the exploration, matrix  $X_{\text{QL}}$  in (3.24) should be rearranged in such a way that linear dependencies between the columns of  $X_{\text{QL}}$  can be used. Write  $\Phi_k$  as:

$$\begin{aligned}\Phi_k &= \phi_k \phi_k^\top - \phi_{k+1} \phi_{k+1}^\top \\ &= \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \begin{bmatrix} \mathbf{x}_k^\top & \mathbf{u}_k^\top \end{bmatrix} - \begin{bmatrix} \mathbf{x}_{k+1} \\ L\mathbf{x}_{k+1} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{k+1}^\top & \mathbf{x}_{k+1}^\top L^\top \end{bmatrix} \\ &= \begin{bmatrix} \Phi_k^{xx} & \Phi_k^{xu} \\ \Phi_k^{ux} & \Phi_k^{uu} \end{bmatrix}\end{aligned}\quad (3.53)$$

with:

$$\begin{aligned}\Phi_k^{xx} &= \mathbf{x}_k \mathbf{x}_k^\top - \mathbf{x}_{k+1} \mathbf{x}_{k+1}^\top \\ \Phi_k^{ux} &= \Phi_k^{xu\top} = L\Phi_k^{xx} + \mathbf{e}_k \mathbf{x}_k^\top \\ \Phi_k^{uu} &= \Phi_k^{ux} L^\top + \mathbf{u}_k \mathbf{e}_k^\top.\end{aligned}\quad (3.54)$$

The rows  $\text{vec}^\triangleleft(\Phi_k)$  of  $X_{\text{QL}}$  do not have the elements arranged according to (3.54), so we redefine  $X_{\text{QL}}$  as:<sup>13 14</sup>

$$X_{\text{QL}} = \begin{bmatrix} \text{vec}^\triangleleft(\Phi_0^{xx})^\top & \text{vec}(\Phi_0^{ux})^\top & \text{vec}^\triangleleft(\Phi_0^{uu})^\top \\ \vdots & \vdots & \vdots \\ \text{vec}^\triangleleft(\Phi_{N-1}^{xx})^\top & \text{vec}(\Phi_{N-1}^{ux})^\top & \text{vec}^\triangleleft(\Phi_{N-1}^{uu})^\top \end{bmatrix} = \begin{bmatrix} \Psi^{xx} & \Psi^{ux} & \Psi^{uu} \end{bmatrix}. \quad (3.55)$$

The submatrices  $\Psi^{xx}$ ,  $\Psi^{ux}$  and  $\Psi^{uu}$  correspond to  $\hat{H}_{xx}$ ,  $\hat{H}_{ux}$  and  $\hat{H}_{uu}$ , which are rearrangements of the vectors  $\hat{\boldsymbol{\theta}}_{xx}$ ,  $\hat{\boldsymbol{\theta}}_{ux}$  and  $\hat{\boldsymbol{\theta}}_{uu}$ . Vector  $\hat{\boldsymbol{\theta}}_{xx}$  has  $n_{xx} = \frac{1}{2}n_x(n_x + 1)$  elements, vector  $\hat{\boldsymbol{\theta}}_{ux}$  has  $n_{ux} = n_u n_x$  elements and vector  $\hat{\boldsymbol{\theta}}_{uu}$  has  $n_{uu} = \frac{1}{2}n_u(n_u + 1)$  elements.

Let  $\mathcal{L}_v$  be the feedback matrix such that  $\text{vec}(L\Phi_k^{xx}) = \mathcal{L}_v \text{vec}(\Phi_k^{xx})$  and let  $\mathcal{L}$  be the feedback such that  $\text{vec}^\triangleleft(\Phi_k^{ux} L^\top) = \text{vec}^\triangleleft(\Phi_k^{ux}) \mathcal{L}^\top$ . Define matrix  $\Upsilon$  with rows  $\text{vec}(\mathbf{e}_k \mathbf{x}_k^\top)$  and matrix  $T$  with rows  $\text{vec}^\triangleleft(\mathbf{u}_k \mathbf{e}_k^\top)$ . Then

$$\Psi^{ux} = \mathcal{L}_v \Psi^{xx} + \Upsilon \quad \text{and} \quad \Psi^{uu} = \Psi^{ux} \mathcal{L}^\top + T \quad (3.56)$$

can be used to find expressions for  $\hat{\boldsymbol{\theta}}_{xx}$ ,  $\hat{\boldsymbol{\theta}}_{ux}$  and  $\hat{\boldsymbol{\theta}}_{uu}$  using (3.33).

<sup>13</sup>For the implementation this is not required, only for the analysis.

<sup>14</sup>The function  $\text{vec}(A)$  stacks all columns of  $A$  into one vector. Note that here  $\text{vec}(\Phi_k^{ux})$  is used instead of  $\text{vec}(\Phi_k^{xu})$ . Only the order of elements is different because  $\Phi_k$  is symmetric and so  $\text{vec}(\Phi_k^{ux}) = \text{vec}(\Phi_k^{xu})^\top$ . The reason for doing this is that the calculation of the feedback according to (3.16) makes use of  $\hat{H}_{ux}$  and not  $\hat{H}_{xu}$ .

## The Feedback

To compute the feedback, (3.16) should be solved using  $\hat{H}_{ux}$  and  $\hat{H}_{uu}$ , so:

$$\hat{L}_{QL} = -\hat{H}_{uu}^{-1}\hat{H}_{ux}. \quad (3.57)$$

$\hat{H}_{uu}$  and  $\hat{H}_{ux}$  are obtained by rearranging the vectors  $\hat{\boldsymbol{\theta}}_{ux}$  and  $\hat{\boldsymbol{\theta}}_{uu}$ . The difference with the SI approach is that the feedback directly follows from the estimations, so it can only be investigated by looking at  $\hat{\boldsymbol{\theta}}_{ux}$  and  $\hat{\boldsymbol{\theta}}_{uu}$ . These estimations are according to (3.25) a solution to:

$$Y_{QL} = \begin{bmatrix} \Psi^{xx} & \Psi^{ux} & \Psi^{uu} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta}_{xx} \\ \boldsymbol{\theta}_{ux} \\ \boldsymbol{\theta}_{uu} \end{bmatrix} + V_{QL}. \quad (3.58)$$

The estimation of  $\hat{\boldsymbol{\theta}}_{uu}$  using (3.33) is given by:

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{uu,i} &= \frac{\Psi_{*i}^{uuT} P_{n_{xx}+n_{ux}+i}^T}{\|P_{n_{xx}+n_{ux}+i} \Psi_{*i}^{uu}\|_2^2} (Y_{QL} - \sum_{j=n_{xx}+n_{ux}+i+1}^{n_{xx}+n_{ux}+n_{uu}} \Psi_{*j}^{uu} \hat{\boldsymbol{\theta}}_{uu,j}) \\ &= \frac{T_{*i}^{eeT} P_{n_{xx}+n_{ux}+i}^T}{\|P_{n_{xx}+n_{ux}+i} T_{*i}^{ee}\|_2^2} (Y_{QL} - \sum_{j=n_{xx}+n_{ux}+i+1}^{n_{xx}+n_{ux}+n_{uu}} \Psi_{*j}^{uu} \hat{\boldsymbol{\theta}}_{uu,j}). \end{aligned} \quad (3.59)$$

$T^{ee}$  has  $\text{vec}^\triangleleft(\mathbf{e}_k \mathbf{e}_k^T)^T$  as rows, because the  $\text{vec}^\triangleleft(L \mathbf{x}_k \mathbf{e}_k^T)^T$  has no effect on the multiplication with matrices  $P_{n_{xx}+n_{ux}+i}$ .<sup>15</sup> Equation (3.59) is similar to (3.35) and without exploration  $T^{ee}$  becomes zero causing a singularity (just like  $\mathcal{E}$  in (3.35)). The estimation of  $\hat{\boldsymbol{\theta}}_{ux}$  has a similar form as (3.36):

$$\hat{\boldsymbol{\theta}}_{ux,i} = \frac{\Upsilon_{*i}^T P_{n_{xx}+i}^T}{\|P_{n_{xx}+i} \Upsilon_{*i}\|_2^2} (Y_{QL} - \sum_{j=n_{xx}+i+1}^{n_{xx}+n_{ux}} \Psi_{*j}^{ux} \hat{\boldsymbol{\theta}}_{ux,j} - \Psi^{uu} \hat{\boldsymbol{\theta}}_{uu}). \quad (3.60)$$

The linear relation  $\Psi^{uu} = \Psi^{ux} \mathcal{L}_v^T + T$  resembles  $\mathcal{U} = \mathcal{X} L^T + \mathcal{E}$ . So it also possible to define a  $\hat{\boldsymbol{\theta}}_d$ , equivalent to the closed loop (3.37) in the SI approach, according to:

$$\hat{\boldsymbol{\theta}}_{d,i} = \frac{\Upsilon_{*i}^T P_{n_{xx}+i}^T}{\|P_{n_{xx}+i} \Upsilon_{*i}\|_2^2} (Y_{QL} - T^{ee} \hat{\boldsymbol{\theta}}_{uu} - \sum_{j=n_{xx}+i+1}^{n_{xx}+n_{ux}} \Psi_{*j}^{ux} \hat{\boldsymbol{\theta}}_{d,j}). \quad (3.61)$$

Since  $\hat{\boldsymbol{\theta}}_d = \hat{\boldsymbol{\theta}}_{ux} + \hat{\boldsymbol{\theta}}_{uu} \mathcal{L}^T$  can be rearranged to  $\hat{H}_d = \hat{H}_{ux} + \hat{H}_{uu} L$ , (3.57) can be written as:

$$\hat{L}_{QL} = \hat{H}_{uu}^{-1}(\hat{H}_{uu} L - \hat{H}_d) = L - \hat{H}_{uu}^{-1} \hat{H}_d. \quad (3.62)$$

With this result two possible outcomes can be given:

- **Too low exploration:**  $\hat{L}_{QL} \approx L$

If the amount of exploration is much too low  $\hat{H}_{uu}$  is much larger than  $\hat{H}_d$ , so the second term in (3.62) can be ignored. The outcome will approximately be the feedback that was used to generate the data!

<sup>15</sup>This is because matrix  $\Phi^{uu}$  is symmetric.

- **High exploration:**  $\hat{L}_{\text{QL}} \approx L'$

For very high exploration, the value of  $V_{\text{QL}}$  in (3.58) can be neglected. So  $\hat{H}$  will be an almost perfect estimation of  $H^L$ . Solving (3.57) will approximately have  $L'$  as outcome.

We can conclude that for insufficient exploration the relative performance does not change. For high amounts of exploration the estimation will almost be correct resulting in a relative performance that corresponds to  $L'$ .

### The Estimation Error

To find the minimal exploration we adapt the expressions for the estimation errors of the SI approach with the values for the Q-Learning approach. The error in the estimation  $\hat{\theta}_{uu}$  is given by:

$$\bar{\theta}_{uu,i} = \frac{T^{ee\text{T}} P^{n_{xx}+n_{ux}+i}}{\|P^{n_{xx}+n_{ux}+i} T^{ee}\|_2^2} (V_{\text{QL}} - \sum_{j=n_{xx}+n_{ux}+i+1}^{n_{xx}+n_{ux}+n_{uu}} T^{ee} \bar{\theta}_{uu,i}). \quad (3.63)$$

In the same way for the estimation  $\hat{\theta}_d$

$$\bar{\theta}_{d,i} = \frac{\Upsilon_{*i}^{\text{T}} P^{n_{xx}+n_{ux}+i}}{\|P^{n_{xx}+n_{ux}+i} \Upsilon_{*i}\|_2^2} (V_{\text{QL}} - T^{ee} \bar{\theta}_{uu} - \sum_{j=n_{xx}+n_{ux}+i+1}^{n_{xx}+n_{ux}+n_{uu}} \Upsilon_{*i} \bar{\theta}_{d,i}). \quad (3.64)$$

The level of magnitude of the errors  $\bar{\theta}_{uu,i}$  and  $\bar{\theta}_{d,i}$  remains the same after rearranging it to  $\bar{H}_{uu}$  and  $\bar{H}_d$ .

### The Minimal Exploration

To get an approximation of the minimal amount of exploration, we start again with some assumptions. Since  $T^{ee}$  has  $\text{vec}^{\leftarrow}(\mathbf{e}_k \mathbf{e}_{ts}^{\text{T}})^{\text{T}}$  as rows, we will assume that  $E\{T_{*i}^{ee\text{T}} T_{*i}^{ee}\} \sim c_1 N \sigma_e^4$ . Using the definition of  $\mathbf{w}_k$  in (3.19) we will assume that  $E\{T_{*i}^{ee\text{T}} V_{\text{QL},*i}\} \sim c_2 N \sigma_e^2 \sigma_v^2$ . We further assume that  $E\{\|P^{n_{xx}+n_{ux}+i} \Upsilon_{*i}\|_2^2\} \sim \sigma_e^2 E\{\|P_i \mathcal{X}_{*i}\|_2^2\}$ , so that the levels of magnitude of the expected errors are:

$$E\{\bar{H}_{uu}\} \sim \frac{c_2 \sigma_v^2}{c_1 \sigma_e^2} \quad (3.65)$$

$$E\{\bar{H}_d\} \sim \frac{\sigma_v^2}{\sqrt{(c_3 + c_4 \sigma_v^2) \sigma_e + c_5 \sigma_v \sigma_e^2 + c_6 \sigma_e^3}}. \quad (3.66)$$

Both errors in (3.66) will be zero if  $\sigma_v = 0$ . This corresponds with the noise free situation in [16] and [44]. In this situation the only purpose of the exploration is to prevent singularity in the computations of the least squares estimate.

The maximum of  $E\{\bar{H}_d\}$  can be expressed as  $\sigma_v(1 + \kappa)$ , where  $\kappa > 0$  is some value that depends on the constants and  $\sigma_v$ . Without specifying the constants it is impossible to get an idea about the size of  $\kappa$ . The only thing we can conclude is:

*The amount of exploration required by the LQRQL approach is larger than the amount of exploration required by the SI approach!*

### 3.3.4 The Exploration Characteristics

Our main contribution in this chapter is comparing the performances for the SI and LQRQL approach. Especially we focused on the influence of the amount of exploration on these performances. We will summarize our results in this section. For this we define the *exploration characteristic* as the expected performance as a function of the amount of exploration  $\sigma_e$ . As performance measure we will use the relative performance introduced in section 3.2.5. First we will give a description of the similarities and then of the differences between the two solution methods.

The outcome of both methods depends on two estimations;  $\hat{B}$  and  $\hat{D}$  for the SI approach,  $\hat{H}_{uu}$  and  $\hat{H}_d$  for LQRQL. These estimations can be viewed as the sum of the correct result and the estimation error (i.e.  $\hat{B} = B + \bar{B}$ ), where the exploration and system noise only affect the estimation error. Based on the influence of the exploration on the estimation errors, we can distinguish four types of outcomes. With the increase of the level of exploration we will have the following types of outcome:

- I **Singularity:** No exploration will result in a singularity, so there is no outcome.
- II **Error Dominance:** If the amount of exploration is much too low, but a feedback can be computed, the estimation errors will dominate the outcome. The resulting feedback will approximately be the feedback that was used to generate the data, so the relative performance does not improve.
- III **Sequence Dependent Outcome:** For a certain amount of exploration, the estimation errors do not dominate the outcome but are still too high to be neglected. So the resulting feedback is partly based on the estimation error and partly on the correct value. The outcome will depend on the particular realization of the exploration noise sequence and system noise sequence. Therefore the relative performance can be anything, although it is bounded from below due to the error  $\bar{D}$  or  $\bar{H}_d$ .
- IV **Correct Estimation:** For sufficient exploration the estimation errors can be neglected. The SI approach will result in  $L^*$  because the system's parameters are estimated correctly. The LQRQL will result in  $L'$  because the parameters of the Q-function are estimated correctly.

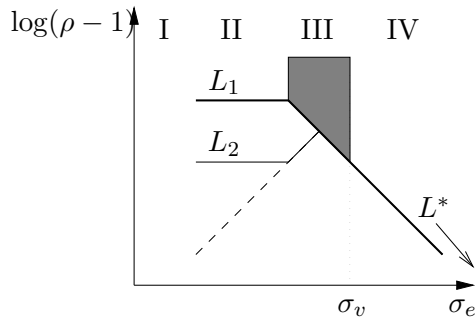
Two exploration characteristics for SI and LQRQL approaches are sketched in figure 3.2. To stress the differences,  $\log(\rho^L - 1)$  is shown instead of  $\rho^L$ . The differences in the four types of outcomes are:

- I **Singularity:** The amount of exploration appears quadratically in the estimations for the QL approach, so it requires more exploration to prevent singularity. This means that the lines in figure 3.2(b) start for higher values of  $\sigma_e$  than in figure 3.2(a).

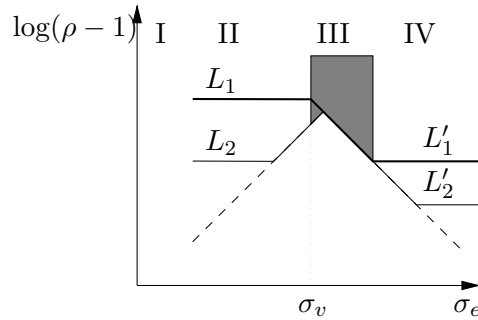
- II **Error Dominance:** Only the value of  $\sigma_e$  for which this is the outcome differs.
- III **Sequence Dependent Outcome:**  $\bar{D}$  has a maximum value for  $\sigma_e < \sigma_v$  and  $\bar{H}_d$  for  $\sigma_e > \sigma_v$ . Also the maximum values are different. So this outcome only differs in the lower bound for the relative performance.
- IV **Correct Estimation:** For the SI approach the relative performance will approach one with the increase in the exploration level. For LQRQL the feedback will approach  $L'$  so that the relative performance depends on the feedback that was used to generate the data. The only way to let the relative performance approach one is to do more policy improvement steps. In Figure 3.2b we see that  $L_2 = L'_1$ , so  $L'_2$  is the result after two policy improvement step when starting with  $L_1$ .

The main differences between the exploration characteristics in figure 3.2 are the amounts of exploration for which these outcomes occur. It is clear that there is a difference in the type IV outcome, because the SI approach will approximate  $L^*$  and the LQRQL approach  $L'$ . Therefore does the outcome of the LQRQL approach depend on the  $L$  and for the SI approach it does not.

The dashed lines in figure 3.2(a) indicate the lower bound on the relative performance due to  $\bar{D}$  (the dashed line continues under the solid and bold line). The relative performance will not go below this line, even when an almost optimal feedback like  $L_2$  is used. So



(a) The SI exploration characteristic. The arrow with  $L^*$  indicates that the characteristic will approach the optimal feedback if the exploration is increased.



(b) The LQRQL exploration characteristic.

**Figure 3.2. The Exploration Characteristics.** Both figures show  $\log(\rho^L - 1)$  as a function of  $\sigma_e$  when data was generated using feedback  $L_1$  (bold line) and feedback  $L_2$  (solid line). The symbols next to these lines indicate the value that is being approximated. The feedback  $L_2$  is almost optimal so that  $\rho^{L_2}$  is almost one. The dashed line indicates the lower bound on the relative performances due to the errors  $\bar{D}$  or  $\bar{H}_d$ . The grey area indicates outcome III, where any relative performance above the lower bound is possible.

taking  $\sigma_e = \sigma_v$  will not “improve” the feedback. (This amount of exploration will give an improvement for  $L_1$ ). Feedback  $L_2$  can only be improved by increasing the amount of exploration even more. Figure 3.2(b) shows the same effect for the LQRQL approach due to  $\bar{H}_d$ . It takes more exploration to guarantee  $L'_2$  than it takes to guarantee  $L'_1$ . Both approaches have in common that for near optimal feedbacks more exploration is required to guarantee an improvement than just avoiding outcome III.

## 3.4 Simulation Experiments

The purpose of the simulation experiments is to verify the results presented in previous sections and show the exploration characteristics.

### 3.4.1 Setup

We take a system according to (3.1) with the following parameters:

$$A = \begin{bmatrix} -0.6 & 0.4 \\ 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \quad (3.67)$$

For the direct cost (3.2) we take  $S$  to be a unit matrix and  $R = 1$ . For this system the number of parameters that has to be estimated for both approaches equals 6. So taking  $N = 20$  measurements should be enough for the estimation. The measurements are generated according to (3.6) with

$$L = \begin{bmatrix} 0.2 & -0.2 \end{bmatrix} \quad \sigma_v = 10^{-4}. \quad (3.68)$$

For this value of  $L$  the closed loop is stable. The solution of the DARE and the optimal feedback are given by:

$$K^* = \begin{bmatrix} 2.302 & 0.149 \\ 0.149 & 1.112 \end{bmatrix} \quad L^* = \begin{bmatrix} 0.373 & 0.279 \end{bmatrix}. \quad (3.69)$$

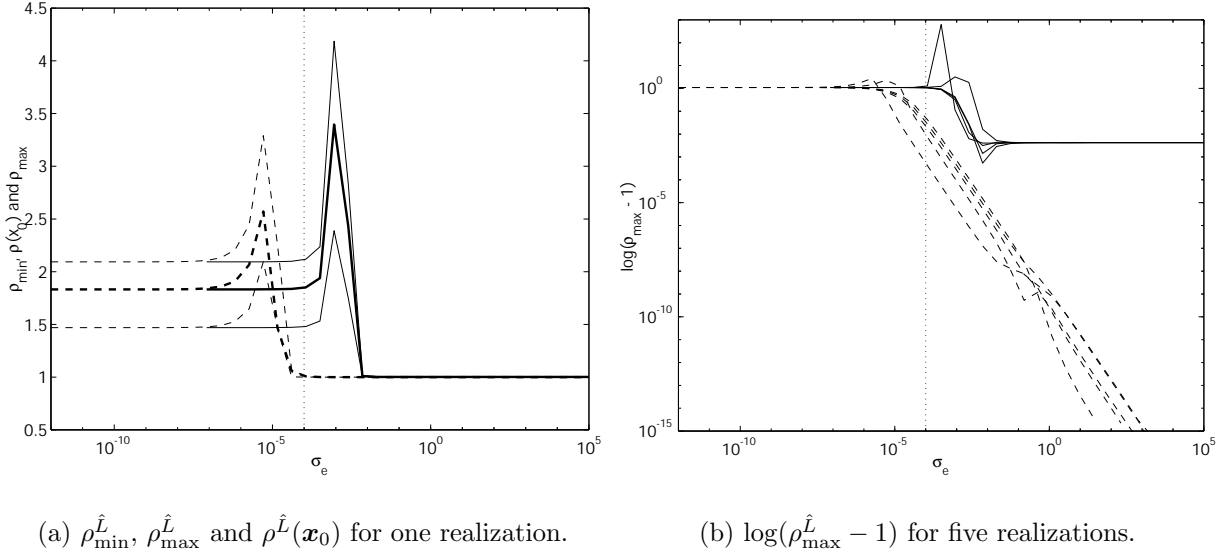
Also the relative performances for  $L$  can be computed:

$$\rho_{\min}^L = 1.469 \quad \rho_{\max}^L = 2.093 \quad \rho^L(\mathbf{x}_0) = 1.832. \quad (3.70)$$

### 3.4.2 Exploration Characteristic

We compute the exploration characteristic by doing the same simulation experiment for different values of  $\sigma_e$ . To make sure that  $\sigma_e$  is the only parameter that differs, we always use the same realizations of exploration noise and system noise by using the same seeds for the random generator. We vary  $\sigma_e$  from  $10^{-12}$  to  $10^5$ .

Figure 3.3(a) shows  $\rho_{\min}^{\hat{L}}$ ,  $\rho_{\max}^{\hat{L}}$  and  $\rho^{\hat{L}}(\mathbf{x}_0)$  for the SI and LQRQL approach for one realization. The exploration intervals for the four types of outcomes can be seen in figure 3.3(a):

(a)  $\rho_{\min}^L$ ,  $\rho_{\max}^L$  and  $\rho^L(\mathbf{x}_0)$  for one realization.(b)  $\log(\rho_{\max}^L - 1)$  for five realizations.

**Figure 3.3. Simulation Results.** The dotted vertical line indicates the system noise level  $\sigma_v = 10^{-4}$ . The dashed lines are the results for the SI approach and the solid lines are the results for the LQRQL approach.

- I SI:  $\sigma_e < 10^{-12}$  (not shown in figure 3.3(a)), LQRQL:  $\sigma_e < 10^{-7}$ .
- II SI:  $10^{-12} < \sigma_e < 10^{-7}$ , LQRQL:  $10^{-7} < \sigma_e < 10^{-4} = \sigma_v$ . The values of  $\rho_{\min}^L$ ,  $\rho_{\max}^L$  and  $\rho^L(\mathbf{x}_0)$  in figure 3.3(a) agree with the values in (3.70).
- III SI:  $10^{-7} < \sigma_e < 10^{-4}$ , LQRQL:  $10^{-4} < \sigma_e < 10^{-2}$ . This particular realization results in an increase of the RP for both methods.
- IV SI:  $\sigma_e > 10^{-4}$ , LQRQL:  $\sigma_e > 10^{-2}$ . The RP for both approaches seem to be one.

In the results in figure 3.3(b) we want to focus on type III and IV outcomes. In figure 3.3(a) the outcomes of type IV are not very clear. Therefore figure 3.3(b) shows  $\log(\rho_{\max}^L - 1)$  and not  $\rho_{\max}^L$ . The outcomes are shown for five different realizations.

- III For some realizations  $\rho_{\max}^L < \rho_{\max}^L$  and for other realizations  $\rho_{\max}^L > \rho_{\max}^L$ . This hold for both approaches but not always for the same realizations (not shown in figure 3.3(b), where the lines are not labeled). So if  $\rho_{\max}^L$  is high for one approach, it does not imply that it is also high for the other approach.
- IV The RP for both approaches are *not* equal to one, they are close to one! For the SI approach the value of  $\rho_{\max}^L$  gets closer to one if the amount of exploration is increased. For LQRQL the value of  $\rho_{\max}^L$  does not approach one if the amount of exploration is increased. Instead it approaches  $\rho_{\max}^{L'} > 1$ .

## 3.5 Discussion

In this chapter we continued investigating RL in the context of LQR as in [16][44]. In order to make it more realistic we included system noise in our analysis. The proofs of convergence in [16][44] are only valid if there is no noise and sufficient exploration is used.

In this chapter we showed that the system noise determines the amount of exploration that is sufficient. There is a hard threshold level for the amount of exploration required. Just below that threshold the resulting feedback can be anything and even result in an unstable closed loop. This is the amount of exploration that has to be avoided. For the SI approach an alternative method was proposed to deal with the bias resulting from the system noise [21]. The idea is to add a bias towards the optimal solution. The effect of such an approach is that this may reduce the probability of an unstable closed loop for the type III outcome, but this can still not be guaranteed. Therefore avoiding type III outcomes is safer.

If we reduce the amount of exploration even more, we will find the feedback used to generate the data as the optimal solution. Although this is not dangerous, this result is not very useful. If the amount of exploration is reduced even more, no feedback can be computed because of a singularity in the least squares estimation. This effect is also present without noise, so the purpose of exploration in [16][44] is to prevent numerical problems with the recursive least squares estimation. The amount of exploration that is sufficient in that case is determined by the machine precision of the computer.

We also compared the result of LQRQL with an indirect approach. We observed that the performance of this approach as a function of the amount of exploration is very similar to that of LQRQL. The main difference is that the threshold level of the amount of exploration required is lower. This means that under the circumstances under which convergence of LQRQL can be proven, it is wiser to use the indirect approach.

In [32] some additional experiments are described. There, two almost identical data sets are shown, where one data set did not change the feedback and where the other gave an improvement. This indicates that visual inspection of the data does not reveal whether sufficient exploration was used. For the LQRQL approach we can look at the eigenvalues of  $\hat{H}$ . If it has negative eigenvalues, the quadratic Q-function is not positive definite and therefore insufficient exploration was used. For the SI approach such an indication is not available.

We did not look at the influence of the feedback itself, but this can only have an effect for the higher amounts of exploration. Just below the threshold level the feedback determines the probability of an unstable closed loop. Since this situation has to be avoided, this is not of interest. For sufficient exploration the feedback will determine how many policy iteration steps are required. When starting with a good performing feedback only a few steps are required.

Our analysis was based on estimating the magnitudes of the estimation errors. These errors still depend on the number of time steps used. The contribution of the number of time steps on the performance for the indirect approach is described in [27]. The results presented there are very conservative and indicate that a large number of time steps are

required for a guaranteed improvement of the performance. Based on our experiment we see that the amount of time steps required is just a couple of times the number of parameters to estimate.

## 3.6 Conclusion

We have shown a fair comparison between two different approaches to optimize the feedback for an unknown linear system. For the system identification approach the estimation and optimization are performed separately. For the Q-Learning approach the optimization is implicitly included in the estimation. The comparison is fair because both approaches used the same data, and no other parameters had to be chosen. So the differences in performance are due to the approaches.

The first conclusion is that for insufficient exploration the result of the optimization will be the same as the feedback that was used to generate the data. So no change in feedback does not imply that the feedback is already optimal. This result is a consequence of the noise in the system. This noise introduces a bias in the estimation and when using insufficient exploration, this bias dominates the estimated outcome.

If the exploration is insufficient, but large enough that the resulting feedback will not be the same as the initial feedback, then the resulting performance becomes very unpredictable. The closed loop can be stable and the performance can be improved, but it is also possible that the closed loop becomes unstable. These results have to be avoided and therefore it is very important that sufficient exploration is used.

The second conclusion is that the LQRQL approach can be guaranteed to optimize the feedback. This is the first continuous state space problem with noise, for which a reinforcement learning approach can be guaranteed to work. This is the good news. The bad news is that if the conditions hold and a good outcome can be guaranteed, an alternative approach based on system identification will perform better. So we can not recommend to use the LQRQL approach for the linear quadratic regularization task.

# Chapter 4

## LQRQL for Nonlinear Systems

### 4.1 Introduction

In this chapter we will study the applicability of linear approximations for nonlinear system. First we will show that other approaches to design controllers for nonlinear systems are often based on local linear approximations. We will rewrite the nonlinear system as a linear system with a nonlinear correction. This allows us to study the effect of nonlinear correction on the estimations of the SI and LQRQL approach as described in chapter 3. We can show that these two approaches estimate the parameters of the wrong function if this correction is not zero.

In a local part of the state space of a smooth nonlinear function, the correction can be assumed to have a constant value. For this situation we will introduce the extended LQRQL approach. In this approach the parameters are estimated of a more general quadratic Q-function, so that more parameters have to be estimated. The resulting feedback function no longer has to go through the origin. Therefore this approach is more suited in a local part of the state space of a nonlinear function.

The effect of the extension of LQRQL is shown in experiments on a nonlinear system. The choice of system was such that we were able to vary the effect of nonlinearity by the choice of the initial state. In this way we were able to show how the extended approach compares with the other two approaches for different sizes of the nonlinear correction. The experiments were performed in simulation and on the real system.

### 4.2 Nonlinearities

#### 4.2.1 The nonlinear system

In this chapter we will only consider systems that can be represented by first order vectorized difference equations. This is the class of time discrete Markov systems. The systems are described as (1.1)

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k), \quad (4.1)$$

where  $\mathbf{f}$  is a functional mapping that maps the present state, control action and noise to the next state value. We will also assume that  $\mathbf{f}$  is a smooth continuous differential mapping, for which the gradient to  $\mathbf{x}_k$  and  $\mathbf{u}_k$  is bounded.

One class of systems that agrees with (4.1) is the class of linear systems according to (3.1). This is an important class. Due to the linearity, the mathematics of analyzing the system's behavior becomes tractable. For a controller design strategy this is important, because this makes it possible to verify whether control design specification are met. Therefore many controller design strategies are based on linear systems.

To illustrate how linear systems simplify the controller design, take a task for which the objective is to keep the state value fixed at a value  $\mathbf{x}_s$ . The state value does not change if  $\mathbf{x}_{k+1}$  equals  $\mathbf{x}_k$ . This means that there has to exist a constant control action  $\mathbf{u}_s$  for which

$$\mathbf{x}_s = A\mathbf{x}_s + B\mathbf{u}_s \quad (4.2)$$

holds. The state that is unchanged under control action  $\mathbf{u}_s$  is given by:

$$\mathbf{x}_s = (I - A)^{-1}B\mathbf{u}_s, \quad (4.3)$$

where we will call  $\mathbf{x}_s$  the set point. This shows that if a solution exists, the state value  $\mathbf{x}_s$  is uniquely determined by the control action  $\mathbf{u}_s$ . It is also important to note that the existence of this solution is completely determined by the parameters of the linear system. So if it exists, it exists for all possible control actions  $\mathbf{u}_s$ .

The result of the design of the controller is that we get a system with a feedback controller. In chapter 1 we already showed that for a linear system with a linear feedback, the behavior of the closed loop can be expressed using the parameters of the closed loop. If the close loop is stable the system will always end up in the origin. If the closed loop is unstable, the system will never end up in the origin.

If we take a general nonlinear function as in (4.1) then instead of (4.2) we have

$$\mathbf{x}_s = \mathbf{f}(\mathbf{x}_s, \mathbf{u}_s, 0). \quad (4.4)$$

In this case it is possible to have more solutions. More solutions means that for one  $\mathbf{x}_s$  more values of  $\mathbf{u}_s$  are possible, but also that for one  $\mathbf{u}_s$  more values for  $\mathbf{x}_s$  are possible. This implies that the correctness of a control action can only be verified when considering the exact value of the state and control action. The same holds for the equilibrium state when a feedback function  $\mathbf{u} = \mathbf{g}(\mathbf{x})$  is used. The equilibrium state is the solution to

$$\mathbf{x}_{\text{eq}} = \mathbf{f}(\mathbf{x}_{\text{eq}}, \mathbf{g}(\mathbf{x}_{\text{eq}}), 0). \quad (4.5)$$

Again it is possible that multiple solutions exist. For some solutions the system will approach the equilibrium while for others it will not. The consequence is that the stability of the system also depends on part of the state space. In some parts of the state space the closed loop can be stable, while in other parts it is unstable. It is clear that this will make the design of a controller very complicated. For the linear case it is just a matter of

making the closed loop stable, for the nonlinear case it may also include defining the part of the state space where the closed loop has to be stable.

We can conclude that the main difference between linear and nonlinear systems is that for linear systems the properties are globally valid. They are given by the parameters of the linear system and do not depend on the state value. For nonlinear systems, properties can be only locally valid. They are given not only by the parameters but also depend on the value of the state.

## 4.2.2 Nonlinear approaches

We will give a short overview of some methods for obtaining feedback functions for nonlinear systems. We will also show how these methods rely on techniques derived for linear systems.

### Fixed Set Point

In (4.4) the noise was set to zero to define the set point  $\mathbf{x}_s$  and the corresponding constant control action  $\mathbf{u}_s$ . In practice however there is always noise. So even if we have a nonlinear system whose state value equals  $\mathbf{x}_s$  when  $\mathbf{u}_s$  is applied, due to the noise the next state value can be different. Then the system is no longer in its set point, so the state value will change again. For a general function  $\mathbf{f}$  this change can be anything and does not have to bring the system back in its set point.

To make sure the system will go back to the set point, a feedback can be added. The input of this feedback is the difference between the state value and the set point. So the task of this feedback is to control its input to zero. To design a linear feedback that will do this, a local linearization of the nonlinear function has to be made around the set point.

If the mapping  $\mathbf{f}$  of the nonlinear system (4.1) is continuously differentiable, the system can be assumed to be locally linear around the set point. Let  $\bar{\mathbf{x}}_k = \mathbf{x}_k - \mathbf{x}_s$  and  $\bar{\mathbf{u}}_k = \mathbf{u}_k - \mathbf{u}_s$ . Then, (4.1) can be rewritten according to:

$$\mathbf{x}_{k+1} - \mathbf{x}_s = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k) - \mathbf{x}_s \quad (4.6)$$

$$\bar{\mathbf{x}}_{k+1} = \mathbf{f}(\mathbf{x}_s + \bar{\mathbf{x}}_k, \mathbf{u}_s + \bar{\mathbf{u}}_k, \mathbf{v}_k) - \mathbf{x}_s \quad (4.7)$$

$$\begin{aligned} &\approx \mathbf{f}(\mathbf{x}_s, \mathbf{u}_s, 0) + \mathbf{f}_x(\mathbf{x}_s, \mathbf{u}_s, 0)\bar{\mathbf{x}}_k + \mathbf{f}_u(\mathbf{x}_s, \mathbf{u}_s, 0)\bar{\mathbf{u}}_k \\ &\quad + \mathbf{f}_v(\mathbf{x}_s, \mathbf{u}_s, 0)\mathbf{v}_k - \mathbf{x}_s \end{aligned} \quad (4.8)$$

$$\bar{\mathbf{x}}_{k+1} = \tilde{A}\bar{\mathbf{x}}_k + \tilde{B}\bar{\mathbf{u}}_k + \tilde{B}_v\mathbf{v}_k. \quad (4.9)$$

In (4.6) the set point  $\mathbf{x}_s$  is subtracted on both sides of (4.1). Then this is expressed in the new state and control action  $\bar{\mathbf{x}}_k$  and  $\bar{\mathbf{u}}_k$  in (4.7). Note that we assume that  $\mathbf{v}$  has zero mean. In (4.8) the mapping  $\mathbf{f}$  is replaced by its first order Taylor expansion around the set point, where the mappings  $\mathbf{f}_x$ ,  $\mathbf{f}_u$  and  $\mathbf{f}_v$  contain the appropriate derivatives. If  $\mathbf{u}_s$  is chosen in such a way that  $\mathbf{f}(\mathbf{x}_s, \mathbf{u}_s, 0) = \mathbf{x}_s$  and the mappings are replaced by the matrices  $\tilde{A}$ ,  $\tilde{B}$  and  $\tilde{B}_v$ , then this forms the linear system (4.9).

In (4.9) we have a linear system for which a linear feedback can be designed, but care has to be taken. The linear system is only valid near the set point so its properties are not globally valid.

### Gain Scheduling

The fixed set point approach only gives a feedback that is valid near the set point. To get a feedback function that is valid in a larger part of the state space, multiple linear models can be used. This means that locally linear models are computed for many different set points. As explained in chapter 2 we can form one global nonlinear feedback by combining all the local linear models.

One approach is to form one global feedback where the control action is determined by only one local linear model that is valid at the current state value. This is Gain Scheduling. A possible implementation of Gain Scheduling is to partition the state space and compute for each partition a local linearized model with respect to the center of the partition. These linear models can be used to compute the appropriate local linear feedbacks. The only difference with the fixed set point approach is that now the local linear feedback does not have to result in an equilibrium state around the center of the partitioning. The feedback may change the state of the system to a part of the state space where a different local model will determine the control action.

### Feedback Linearization

In some cases it is possible to change a nonlinear system into a globally linear system. This is called feedback linearization. The objective is to change the nonlinear system into a linear system of our choice. This is only possible for a restricted class of nonlinear systems for which the mapping  $\mathbf{f}$  of the nonlinear system in (4.1) can be written as

$$\mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \mathbf{g}(\mathbf{x}) + \mathbf{h}(\mathbf{x})\mathbf{u} + \mathbf{v}, \quad (4.10)$$

with  $\mathbf{g}$  and  $\mathbf{h}$  mappings of the appropriate dimensions.

Suppose we want to have a linear system with parameters  $\tilde{A}$  and  $\tilde{B}$  and control input  $\tilde{\mathbf{u}}$ . Then the control action can be computed according to

$$\mathbf{u} = \mathbf{h}(\mathbf{x})^{-1}(\tilde{A}\mathbf{x} - \tilde{B}\tilde{\mathbf{u}} - \mathbf{g}(\mathbf{x})), \quad (4.11)$$

where  $\tilde{\mathbf{u}}$  represents a new control action. Applying (4.11) to a nonlinear system with (4.10) will transform the nonlinear system into a linear system:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{g}(\mathbf{x}_k) + \mathbf{h}(\mathbf{x}_k)\mathbf{u}_k + \mathbf{v}_k \\ &= \mathbf{g}(\mathbf{x}_k) + \mathbf{h}(\mathbf{x}_k)\mathbf{h}(\mathbf{x}_k)^{-1}(\tilde{A}\mathbf{x}_k - \tilde{B}\tilde{\mathbf{u}}_k - \mathbf{g}(\mathbf{x}_k)) + \mathbf{v}_k \\ &= \tilde{A}\mathbf{x}_k - \tilde{B}\tilde{\mathbf{u}}_k + \mathbf{v}_k. \end{aligned} \quad (4.12)$$

The result is a linear system, so the appropriate new control actions  $\tilde{\mathbf{u}}$  can be determined using conventional linear techniques. The choice of the parameters of the new linear system are made to simplify the design of the controller for the new linear system.

It is clear that this approach is only possible for a restricted class of nonlinear systems. Not only does (4.10) have to hold, but also (4.11) has to exist. In case of a scalar state and action it is clear that  $\mathbf{h}(x) \neq 0$  is a sufficient condition for the existence of (4.11). For more general situations the existence has to be verified using Lie-brackets. For more detailed information about this approach see [37].

The existence of (4.11) is important, but it also has to be found. When the physics of the system is known this can be computed. When the physics is not known (4.11) has to be approximated by a general function approximator. In that case it is no longer possible to guarantee global properties of the system, since it relies on the quality of the approximation. For a continuous time configuration, conditions for sufficient quality of the approximation for feedback linearization can be given [78]. One of these conditions is that the system is excited enough, so that the state space is explored sufficiently.

### Linear Matrix Inequalities

Linear Matrix Inequalities (LMI) techniques [15] are used to analyze nonlinear systems and proof properties, like stability. These techniques are based on the idea that a nonlinear system can be described by a set of linear systems. If we look at only one state transition of a nonlinear system then a linear system can be defined that generates the same state transition:<sup>1</sup>

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = A(\mathbf{x}_k, \mathbf{u}_k)\mathbf{x}_k + B(\mathbf{x}_k, \mathbf{u}_k)\mathbf{u}_k. \quad (4.13)$$

Here  $A(\mathbf{x}_k, \mathbf{u}_k)$  and  $B(\mathbf{x}_k, \mathbf{u}_k)$  represent matrices, whose parameters depend on  $\mathbf{x}_k$  and  $\mathbf{u}_k$ . So for the current state and control action there is a linear system for which the state transition will also lead to  $\mathbf{x}_{k+1}$ .

To say something about the stability of the nonlinear system, all possible values of  $\mathbf{x}$  and  $\mathbf{u}$  should be considered. The parameter matrices  $A(\mathbf{x}, \mathbf{u})$  and  $B(\mathbf{x}, \mathbf{u})$  for all possible  $\mathbf{x}$  and  $\mathbf{u}$  form a set. If all linear systems corresponding to the matrices in the set are stable, then also the nonlinear system must be stable. This is because for any state transition of the nonlinear system there exists a stable linear system that will have the same state transition.

The set can consist of an infinite number of linear systems, making it impossible to prove the stability of all linear systems. One solution is to use a polytope LMI. In the parameter space of the linear system a polytope is selected that encloses the complete set. If the linear systems corresponding to the edges of the polytope are stable then all linear systems within the polytope are stable as well.

Another approach is to use a norm bounded LMI. One stable linear system with parameters  $\tilde{A}$  and  $\tilde{B}$  is selected and all linear systems are considered to have parameters  $\tilde{A}$  and  $\tilde{B}$  plus an extra feedback. The state transitions of the linear systems are described as  $\mathbf{x}_{k+1} = \tilde{A}\mathbf{x}_k + \tilde{B}\mathbf{u}_k + \mathbf{w}(\mathbf{x}_k, \mathbf{u}_k)$ , where the vector  $\mathbf{w}(\mathbf{x}_k, \mathbf{u}_k)$  represents the extra feedback.

---

<sup>1</sup>For simplicity we assume there is no noise.

If it can be proven that the norm of  $\mathbf{w}$  is less than one for all values of  $\mathbf{x}$  and  $\mathbf{u}$ , then the nonlinear system is stable.

Since LMIs provide the opportunity to guarantee stability for nonlinear systems, they can be used for stability based controller design. The LMI technique is used to specify the set of feasible feedbacks and a controller design approach is used to select one feedback from that set. The drawback of the LMI approaches is that the set of feasible feedbacks depends on the choice of polytope or  $\tilde{A}$  and  $\tilde{B}$ . In case of an unfortunate choice it is possible that many good feedbacks are rejected. In the worst case the set of feasible feedbacks is empty.

### 4.2.3 Summary

We described some approaches to obtain feedback functions for nonlinear systems. Except for the feedback linearization, all were based on local linear approximations of the nonlinear feedback. These approaches assume that the mapping  $\mathbf{f}$  is available, but when it is unknown the local linear system can be approximated by the SI approach from chapter 3. The LQRQL approach in chapter 3 was also derived for linear systems. We are interested in using LQRQL to obtain a local linear approximation of the optimal feedback function.

## 4.3 The Extended LQRQL Approach

In this section we will first show that the SI and LQRQL approach from chapter 3, do have limitations when they are applied to nonlinear systems. This is especially the case when the estimations are based on data generated in a small part of the state space. We will show that LQRQL can be extended to overcome these limitations, resulting in a solution that is more appropriate in a small part of the state space.

### 4.3.1 SI and LQRQL for nonlinear systems

The SI and LQRQL approach were derived for linear systems. In order to see how they would perform when applied to nonlinear systems, we write (4.1) as

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k) \quad (4.14)$$

$$\begin{aligned} &= \tilde{A}\mathbf{x}_k + \tilde{B}\mathbf{u}_k + (\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k) - \tilde{A}\mathbf{x}_k - \tilde{B}\mathbf{u}_k) \\ &= \tilde{A}\mathbf{x}_k + \tilde{B}\mathbf{u}_k + \mathbf{w}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k). \end{aligned} \quad (4.15)$$

This describes a linear system with parameters  $\tilde{A}$  and  $\tilde{B}$  and an extra additional nonlinear correction function  $\mathbf{w}$ .<sup>2</sup> Equation (4.15) strongly resembles a norm bounded LMI. In our case the mapping  $\mathbf{f}$  is unknown, so we cannot use (4.15) as a LMI.

The SI and LQRQL approach obtain a feedback based on a generated train set. If we assume that the train set is generated in a small local part of the state space, we can

---

<sup>2</sup>The linear system (3.1) is a special case for which  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \mathbf{v} \forall \mathbf{x}, \mathbf{u}, \mathbf{v}$ .

simplify (4.15) by replacing the nonlinear function  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$  with its average value for the train set. So we look at the system:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{w} \quad (4.16)$$

where the value of vector  $\mathbf{w}$  has a fixed constant value.<sup>3</sup>

We can apply the SI approach, which will result in the estimated linear system:  $\mathbf{x}_{k+1} = \hat{A}\mathbf{x}_k + \hat{B}\mathbf{u}_k$ . In this linear system the correction is not present. This implies that  $\hat{A}$  and  $\hat{B}$  are estimated such that the average value of  $\mathbf{w}$  is zero. So when the average value of  $\mathbf{w}$  is far from zero, the estimated linear system with  $\hat{A}$  and  $\hat{B}$  is *not* a good approximation of  $\mathbf{f}$  in (4.14). The resulting feedback  $\hat{L}_{\text{SI}}$  will not approximate the optimal feedback at the part of the state-action space where the train set was generated.

In order to see the consequences of  $\mathbf{w}$  on the result of LQRQL, we can look at the resulting linear feedback function  $\mathbf{u} = L\mathbf{x}$ . When we apply the feedback function  $\mathbf{u}_k = L\mathbf{x}_k$  to control (4.16), the equilibrium state  $\mathbf{x}_{\text{eq}}$  is no longer at the origin. The equilibrium state is given by:

$$\mathbf{x}_{\text{eq}} = (I - A - BL)^{-1}\mathbf{w}. \quad (4.17)$$

This shows that the equilibrium state depends on  $L$ ! The LQRQL approach was derived for a linear system with  $\mathbf{w} = \mathbf{0}$ , so the equilibrium state was always in the origin. Only the trajectory towards the equilibrium state can be optimized.

At the equilibrium state in (4.17) the direct costs (3.2) are not zero. So LQRQL applied to (4.16) will not only optimize the trajectory toward the equilibrium state, but also the value of the equilibrium state. If both the equilibrium state and the trajectory towards the equilibrium state have to be optimized, then the feedback function  $\mathbf{u} = L\mathbf{x}$  has insufficient degrees of freedom.

### 4.3.2 The extension to LQRQL

Another way to see why the linear feedback does not have enough degrees of freedom is by looking at its approximation of a nonlinear optimal feedback. Let us assume an optimal linear feedback function  $\mathbf{g}^*$  as shown in figure 4.1. The local linear approximation according to  $\mathbf{u} = L\mathbf{x}$  is not very good. The figure also shows that the addition of an extra constant  $l$  to the feedback function will allow for a better local linear approximation.

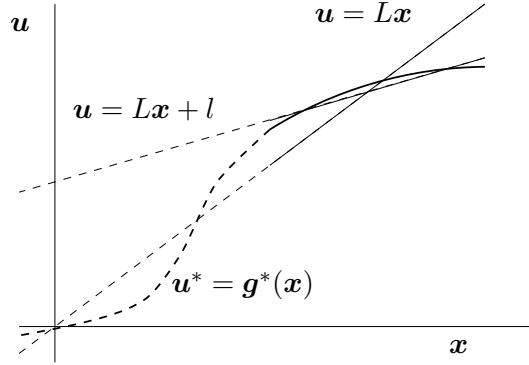
If the feedback function  $\mathbf{u} = L\mathbf{x} + l$  is used for (4.16) then the equilibrium state is given by

$$\mathbf{x}_{\text{eq}} = (I - A + BL)^{-1}(Bl + \mathbf{w}). \quad (4.18)$$

Here we see that if the feedback  $L$  is chosen such that the trajectory towards the equilibrium state is optimal, the value of  $l$  can be used to select the optimal equilibrium state.<sup>4</sup> The  $l$

<sup>3</sup>Strictly speaking this does not have to represent a nonlinear system because the value of  $\mathbf{w}$  is also globally valid. We can also view (4.16) as a linear system, where the mean of system noise is not zero. The  $\mathbf{w}$  then represents the mean of the system noise.

<sup>4</sup>Whether this is possible depends on  $B$  and  $\mathbf{w}$ .



**Figure 4.1.** The local linear approximation of the optimal nonlinear feedback function. The solid lines indicate the region of approximation. Because the feedback  $\mathbf{u} = L\mathbf{x}$  has to go through the origin, the local linear approximation cannot be a good approximation of the nonlinear function. The approximation of  $\mathbf{u} = L\mathbf{x} + l$  does not have to go through the origin and is able to match the nonlinear function more closely.

can be interpreted as the constant  $\mathbf{u}_s$  in the Fixed Set Point approach. The difference is that the value of  $l$  is not chosen, but optimized using the extended LQRQL approach.

The resulting feedback function of LQRQL is a consequence of the choice of the quadratic Q-function. For the new feedback function, a new Q-function is needed. The Q-function used in the previous chapter was of the form  $Q(\phi) = \phi^T H \phi$ , where  $\phi_k^T = [\mathbf{x}_k^T \quad \mathbf{u}_k^T]$ . This is not a very general quadratic function. A general quadratic function is given by:

$$Q(\phi) = \phi^T H \phi + G^T \phi + c. \quad (4.19)$$

By including a term with vector  $G$  and a constant  $c$ , any quadratic function can be represented by (4.19). If (4.19) has the optimal parameters  $H^*$ ,  $G^*$  and  $c^*$ , the greedy feedback can be found by taking the derivative to  $\mathbf{u}^*$  and set this to zero. This results in

$$\mathbf{u}^* = -(H_{uu}^*)^{-1}(H_{xu}^* \mathbf{x} + G^{*\top}) \quad (4.20)$$

$$= L^* \mathbf{x} + l^*. \quad (4.21)$$

This indicates that the Q-function (4.19) will result in the feedback we want. Compared to (3.16) we see that again  $L^* = -(H_{uu}^*)^{-1}H_{xu}^*$ . This shows that  $L^*$  optimizes the trajectory to the equilibrium state. The difference with (3.16) is that a constant  $l^* = -(H_{uu}^*)^{-1}G^{*\top}$  is added to the feedback function. The purpose of this constant is to determine the optimal equilibrium state. Note that the scalar  $c$  in (4.19) does not appear in (4.21), so it does not have to be included in the estimation.

### 4.3.3 Estimating the new quadratic Q-function

The estimation method for the new quadratic Q-function will be slightly different because more parameters have to be estimated. In the previous chapter the estimation was based

on the temporal difference (3.18). A similar approach will be used for the Q-function according to (4.19).<sup>5</sup> The difference with (3.19) is that  $\phi_{k+1}^T = [\mathbf{x}_{k+1}^T \quad L\mathbf{x}_{k+1}^T + l]$  and the consequences of the noise are represented by  $\nu_k$ . Then in the same way as in (3.20):

$$r_k = \sum_{i=k}^{\infty} r_i = Q(\mathbf{x}_k, \mathbf{u}_k) - Q(\mathbf{x}_{k+1}, L\mathbf{x}_{k+1} + l) \quad (4.22)$$

$$= \phi_k^T H \phi_k + G^T \phi_k - \phi_{k+1}^T H \phi_{k+1} - G^T \phi_{k+1} + \nu_k \quad (4.23)$$

$$= \text{vec}^{\triangleleft}(\phi_k \phi_k^T) \text{vec}^{\triangleleft}(H) - \text{vec}^{\triangleleft}(\phi_{k+1} \phi_{k+1}^T) \text{vec}^{\triangleleft}(H) + (\phi_k^T - \phi_{k+1}^T)G + \nu_k \quad (4.24)$$

$$= \begin{bmatrix} \text{vec}^{\triangleleft}(\phi_k \phi_k^T - \phi_{k+1} \phi_{k+1}^T) & \phi_k^T - \phi_{k+1}^T \end{bmatrix} \begin{bmatrix} \text{vec}^{\triangleleft}(H) \\ G \end{bmatrix} + \nu_k. \quad (4.25)$$

This again can be used to express the estimation as a linear least squares estimation:

$$Y_{\text{EX}} = \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{N-1} \end{bmatrix} = \begin{bmatrix} \text{vec}^{\triangleleft}(\phi_0 \phi_0^T - \phi_1 \phi_1^T) & \phi_0^T - \phi_1^T \\ \text{vec}^{\triangleleft}(\phi_1 \phi_1^T - \phi_2 \phi_2^T) & \phi_1^T - \phi_2^T \\ \vdots & \vdots \\ \text{vec}^{\triangleleft}(\phi_{N-1} \phi_{N-1}^T - \phi_N \phi_N^T) & \phi_{N-1}^T - \phi_N^T \end{bmatrix} \theta_{\text{EX}} + \begin{bmatrix} \nu_0 \\ \nu_1 \\ \vdots \\ \nu_{N-1} \end{bmatrix} \quad (4.26)$$

$$= X_{\text{EX}} \theta_{\text{EX}} + V_{\text{EX}}. \quad (4.27)$$

The estimation  $\hat{\theta}_{\text{EX}} = (X_{\text{EX}}^T X_{\text{EX}})^{-1} X_{\text{EX}}^T Y_{\text{EX}}$  gives an estimation of  $\theta_{\text{EX}}$  and because  $\text{vec}^{\triangleleft}(H)$  and  $G$  are included in  $\theta_{\text{EX}}$ , it also gives the estimation for  $\hat{H}$  and  $\hat{G}$ . The  $\hat{H}$  and  $\hat{G}$  can be used in the same way as in (4.21) to obtain the resulting  $\hat{L}_{\text{EX}}$  and  $\hat{l}$ .

The absence of the constant  $c$  in (4.25) indicates that  $c$  does not influence the outcome of the estimation.<sup>6</sup> So the actual function that is estimated is

$$\hat{Q}(\phi) = \phi^T \hat{H} \phi + \hat{G}^T \phi. \quad (4.28)$$

This is the function we will use as Q-function and we will call this the Extended LQRQL approach. For this function  $\hat{Q}(\mathbf{0}) = \mathbf{0}$ , so it is not a general quadratic function anymore. Still this is general enough because the value of  $c$  also does not influence (4.21). The reason that this constant  $c$  can be ignored completely is that it represents the costs that will be received anyway, regardless the optimization. This indicates that the optimization is only based on avoidable costs. Costs that will be received anyway do not influence the resulting feedback function.

## 4.4 Exploration Characteristic for Extended LQRQL

The estimation (4.27) shows how the parameters are estimated, but does not indicate how well these parameters are estimated. The results from the previous chapter suggest that the correctness of the estimation depends on the amount of exploration used for generating

<sup>5</sup>The Q-function depends on  $L$  and  $l$ , but for clarity these indices are omitted.

<sup>6</sup>Note that if a discount factor  $\gamma < 1$  is used in (4.22), the constant  $c$  does influence the outcome.

the train set. The exploration characteristic was introduced in section 3.3.4 to present an overview of the quality of the outcome as a function of the exploration. This will now be used to investigate the results of the extended LQRQL approach. Two questions have to be answered:

- Are the extra parameters  $\hat{G}$  estimated correctly?
- How does the estimation of  $\hat{G}$  influence the estimation of  $\hat{H}$ ?

This last question is relevant since  $G$  and  $H$  are estimated simultaneously in (4.27).

The estimation for the extended approach includes  $n_x$  extra parameters and can be written similar to (3.58). The linear equation that has to hold is given by

$$Y_{\text{EX}} = \begin{bmatrix} \Psi^{xx} & \Psi^{ux} & \Psi^{uu} & \Psi^g \end{bmatrix} \begin{bmatrix} \theta_{xx} \\ \theta_{ux} \\ \theta_{uu} \\ \theta_g \end{bmatrix} + V_{\text{EX}}, \quad (4.29)$$

where  $V_{\text{EX}}$  represents the noise and  $\theta_g$  represents the extra parameters that have to be estimated. Similar to (3.63) the estimation error of a row of  $\hat{\theta}_g$  can be written as

$$\bar{\theta}_{g,i} = \frac{\Psi_{*i}^g \text{T} P^{\text{T}}}{\|P_{n_{xx}+n_{xu}+n_{uu}+i} \Psi_{*i}^g\|_2^2} (V_{\text{EX}} - \sum_{j=n_{xx}+n_{xu}+n_{uu}+1+i}^{n_{xx}+n_{xu}+n_{uu}+n_x} \Psi_{*j}^g \bar{\theta}_{g,j}). \quad (4.30)$$

The estimations  $\bar{\theta}_{uu}$  in (3.63) and  $\bar{\theta}_{ux}$  change according to

$$\bar{\theta}_{uu,i} = \frac{T_{*i}^{ee} \text{T} P^{\text{T}}}{\|P_{n_{xx}+n_{xu}+i} T_{*i}^{ee}\|_2^2} (V_{\text{EX}} - \Psi^g \bar{\theta}_g - \sum_{j=n_{xx}+n_{xu}+1+i}^{n_{xx}+n_{xu}+n_{uu}} \Psi_{*j}^{uu} \bar{\theta}_{uu,j}), \quad (4.31)$$

$$\bar{\theta}_{ux,i} = \frac{\Upsilon_{*i}^{\text{T}} P^{\text{T}}}{\|P_{n_{xx}+i} \Upsilon_{*i}\|_2^2} (V_{\text{EX}} - \Psi^g \bar{\theta}_g - \Psi^{uu} \bar{\theta}_{uu} - \sum_{j=n_{xx}+1+i}^{n_{xx}+n_{xu}} \Psi_{*j}^{ux} \bar{\theta}_{ux,j}). \quad (4.32)$$

The main difference here is that there is the extra  $\Psi^g \bar{\theta}_g$  in the estimation errors  $\bar{\theta}_{uu,i}$  and  $\bar{\theta}_{ux,i}$ . The dependency of the estimation error  $\bar{\theta}_g$  on the exploration is comparable to that of the SI approach, so the influence of the  $\bar{\theta}_{uu,i}$  and  $\bar{\theta}_{ux,i}$  can be neglected. The  $V_{\text{EX}}$  is different from  $V_{\text{QL}}$  from the previous chapter in the sense that it now includes the value of  $\mathbf{w}$ . This means that in the expected estimation errors (3.66) the value of  $\sigma_e^2$  should be replaced by  $(\sigma_e + \|\mathbf{w}\|)^2$ . The consequence is that even for low noise in the system, if there is a large extra  $\mathbf{w}$ , still a large amount of exploration is required. In other words, the minimum of the Q-function should be included in the area that is being explored.

### Simulation of the Extended LQRQL characteristics

We did some simulation experiments to show the reliability of the extended LQRQL approach. To be able to show the correctness of the parameter estimation we took as system the model given by (4.16):

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{v}_k + \mathbf{w} \quad \mathbf{u}_k = L\mathbf{x}_k + \mathbf{e}_k + l. \quad (4.33)$$

In (4.33) the  $\mathbf{w}$  is constant so it is the same for all time steps. In the experiments we compared the results with the standard LQRQL approach.

We first did an experiment to see whether the correct value of  $l$  can be found. Because all parameters are estimated simultaneously, we also did an experiment to see whether the estimation of  $G$  has any positive effects on the estimation of  $H$ . Finally we did an experiment to see whether an arbitrary set point can be learned, when this set point is determined by the direct costs  $r$ .

We used the same parameter settings as in the previous chapter:

$$A = \begin{bmatrix} -0.6 & 0.4 \\ 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad L = \begin{bmatrix} 0.2 & -0.2 \end{bmatrix} \quad \sigma_v = 10^{-4}. \quad (4.34)$$

The value of  $\mathbf{w}$  depends on the experiment. Each experiment was performed 5 times and we took  $N = 30$  number of time steps.

**Experiment I:** *Can a correct value of  $l$  be found?* This requires knowledge about the correct value  $l$ . In this experiment we took  $\mathbf{w}$  equal to zero so that no additional  $l$  is required. We took as initial value  $l = 1$ . The initial value  $l \neq 0$ , so that the estimated  $\hat{l}$  should be zero.

Figure 4.2(a) shows how the value of  $l$  changes. For low values of  $\sigma_e$  the value of  $l$  does not change. If the value of  $\sigma_e$  is larger than  $\sigma_v$  the correct value of  $l$  is obtained.

**Experiment II:** *Does the extension of LQRQL improve the results?* In this experiment we used  $\mathbf{w}^T = [1 \ 1]$ , so that a  $l \neq 0$  was required. Both the LQRQL and the Extended LQRQL were applied on the same train sets. We asked the following questions: Does the nonzero  $\mathbf{w}$  influence the estimation of  $\hat{L}$ ? Also does the nonzero  $\mathbf{w}$  influence the amount of exploration that is required?

Figure 4.2(b) shows  $\|\hat{L} - L^*\|$  for both approaches, where  $L^*$  is the optimal feedback computed with (3.5) and (3.4). For low values of  $\sigma_e$  the value of  $\hat{L}$  equals  $L$  in (4.34). The value of  $\|\hat{L} - L^*\|$  decreased around  $\sigma_e \approx 1$ , which is about the same scale as  $\mathbf{w}$ . We did a similar experiment using  $\mathbf{w}^T = [10 \ 10]$ , and there this happened at  $\sigma_e \approx 10$ . This indicates that the sufficient amount of exploration depends on  $\|\mathbf{w}\|$  if  $\|\mathbf{w}\| > \sigma_v$ . Figure 4.2(b) shows that the improvement for the extended approach requires less exploration. For high values of  $\sigma_e$  the value of  $l$  becomes  $-0.5$  and this makes the total costs for the extended LQRQL approach lower than the total costs of the standard LQRQL approach.

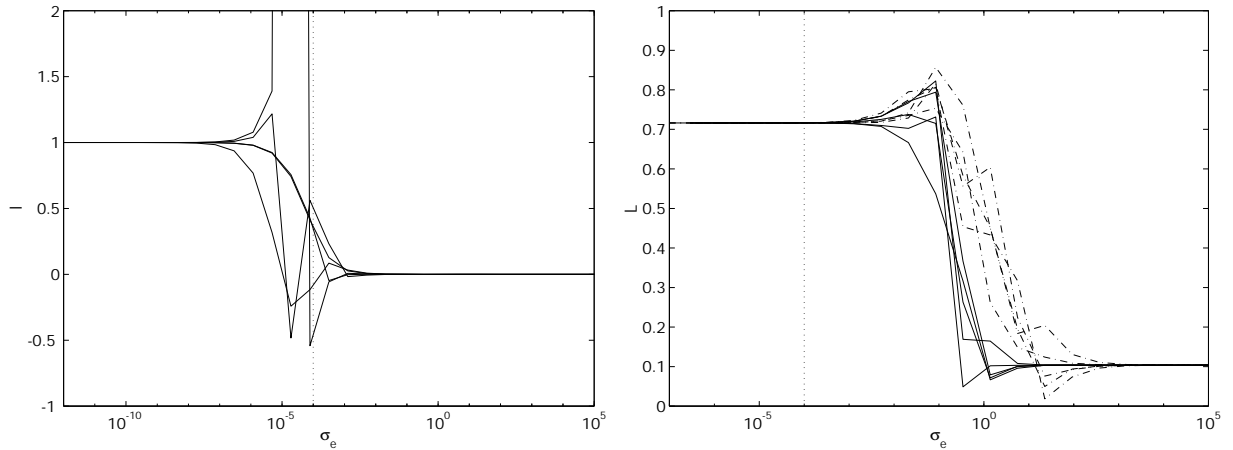
**Experiment III:** *Can the set point be learned?* The set point was introduced in section 4.2.2 to indicate a desired equilibrium state. So learning the set point means optimizing the stationary behavior of the system. The extension of LQRQL was motivated by the inability of LQRQL to deal with the simultaneous optimization of the transient and stationary behavior of the system. We did this experiment to see whether this is possible using the extended LQRQL. We changed the costs in (3.2)

to  $r_k = (\mathbf{x}_k^T - \mathbf{x}_s^T)S(\mathbf{x}_k - \mathbf{x}_s) + \mathbf{u}_k^T R \mathbf{u}_k$ , to specify a preference for an equilibrium point at  $\mathbf{x}_s$ .

Figure 4.3(a) shows the total costs when  $\mathbf{x}_s^T = [1 \ -1]$ . It is clear that the cost reduction after  $\sigma_e \approx 1$  is much larger for the extended approach. The  $\sigma_e \approx 1$  is the same scale as  $\mathbf{x}_s$  and doing the same experiment for a larger  $\mathbf{x}_s$  shows that more exploration is required (not shown).

For one result obtained using  $\sigma_e = 5.6$  the state values are shown as a function of time in figure 4.3(b). The values for the standard approaches always go to zero while the values for the extended approach go to two different values. The value of  $\hat{l} = -0.55$  brings the equilibrium state closer to  $\mathbf{x}_s$ . Note that the equilibrium state will not be equal to  $\mathbf{x}_s$  because of the costs assigned to the control action that keeps it at  $\mathbf{x}_s$ .

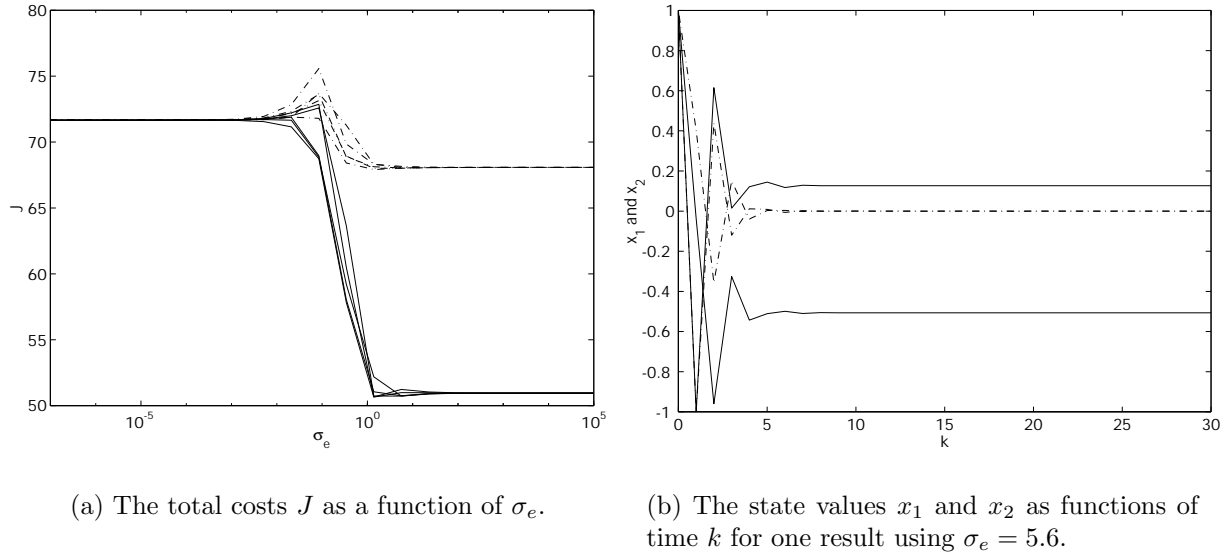
The simulation experiments have shown that the correct value of  $l$  will be found. They have shown that in case of  $\mathbf{w} \neq \mathbf{0}$ , the resulting feedbacks  $\hat{L}_{QL}$  and  $\hat{L}_{EX}$  will have the same outcome. However, the extended approach requires less exploration to obtain this result. Finally the experiments showed that the extended approach is also able to learn the set point.



(a) Experiment I:  $\hat{l}$  as a function of  $\sigma_e$ .

(b) Experiment II:  $\|\hat{L} - L^*\|^2$  as a function of  $\sigma_e$ .

**Figure 4.2.** Results of the simulation experiments I and II. The solid lines are the results for the extended approach, the dashed lines the result of the standard approach. The amount of system noise  $\sigma_v = 10^{-4}$  is indicated by the dotted line.



**Figure 4.3.** Results of simulation experiment III. The solid lines are the results for the extended approach, the dashed lines the result of the standard approach.

## 4.5 Simulation Experiments with a Nonlinear System

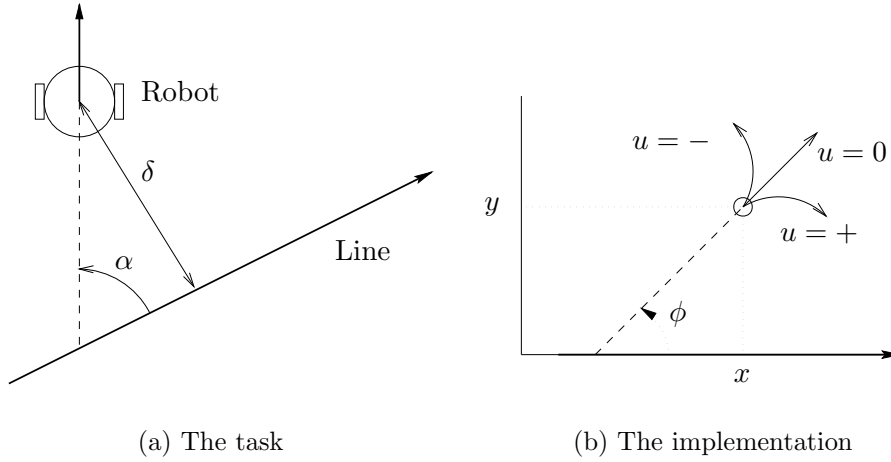
For a nonlinear system (4.15) the value of  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$  does not have to be constant. In the previous section we showed that the extended LQRQL approach performs better than the standard approach when a *constant*  $\mathbf{w}$  is not zero. If the nonlinearity is smooth then in a small part of the state space the value of  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$  does not vary too much. If the average value of  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$  is not zero, the extended LQRQL approach should also perform better. To verify this we did some experiments with a nonlinear system, where we compared the performance of the extended LQRQL approach with the SI and standard LQRQL approach from chapter 3.

### 4.5.1 The nonlinear system

#### The mobile robot

As nonlinear system we used a mobile robot system. A description of the robot can be found in appendix B. Further a model is given in appendix B, that describes how the robot changes its position and orientation in the world. The change of orientation does not depend on the position, but the change in position depends on the orientation. This suggests that this can be used as a nonlinear system, where the effect of the orientation on the change of position introduces the nonlinearity.

The task of the robot is to follow a straight line that is defined in the world. The state is given by the distance  $\delta$  to that line and the orientation  $\alpha$  with respect to that line, as



**Figure 4.4.** The top view of the robot. The left figure illustrates the general task. The right figure shows our implementation where the line to follow is the positive x-axis. This figure also indicates the robot's movement, given a positive, negative or zero  $u$ .

shown in figure 4.4(a). So the state is given by  $\mathbf{x}^T = [\alpha \ \delta]$ , where  $\delta$  is given in meters and  $\alpha$  in radians. Using (B.6) we can express the state transition for the robot given this task:

$$\alpha_{k+1} = \alpha_k + \omega T \quad (4.35)$$

$$\delta_{k+1} = \delta_k + \frac{v_t}{\omega} (\cos(\alpha_k) - \cos(\alpha_k + T\omega)). \quad (4.36)$$

The  $T$  is the sample time. According to (4.36) the trajectory of the robot describes a part of a circle with radius  $\frac{v_t}{\omega}$ . There are two control actions: the traversal speed  $v_t$  and the rotation speed  $\omega$ . We gave the traversal speed a fixed value of 0.1 meters per second. As control action  $u$  we took the rotation speed  $\omega$ . Without loss of generality we can take the x-axis of the world as the line to follow. Our simulation setup is shown in figure 4.4(b), where we take  $\delta = y$  and  $\alpha = \phi$ .

We used quadratic direct costs, so

$$r_k = \mathbf{x}^T \begin{bmatrix} S_\alpha & 0 \\ 0 & S_\delta \end{bmatrix} \mathbf{x} + uRu, \quad (4.37)$$

with

$$S_\alpha = 0.1, \quad S_\delta = 1 \quad \text{and} \quad R = 1. \quad (4.38)$$

This indicates that we want to minimize the distance to the line without steering too much. The  $S_\alpha$  is not so important. The only reason to give it a small positive value is to assign costs to following the line in the wrong direction.

### Comparing the results

Before starting the experiments we will give an indication about the correct feedbacks. We will also show that the value of  $\delta$  can be used to vary the value of  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$ . To compare the results of the three approaches we have to test the performance. We will describe how we will test the results.

The task is to find a linear feedback  $\hat{L}$  by applying the SI approach and both LQRQL approaches. The extended LQRQL approach also has to find an extra term  $\hat{l}$  from the train set. So the resulting feedback for the SI and standard LQRQL approach is:

$$u = \hat{L}\mathbf{x} = \begin{bmatrix} \hat{L}_\alpha & \hat{L}_\delta \end{bmatrix} \begin{bmatrix} \alpha \\ \delta \end{bmatrix}. \quad (4.39)$$

For the extended LQRQL approach there is an extra term

$$u = \hat{L}\mathbf{x} + \hat{l} = \begin{bmatrix} \hat{L}_\alpha & \hat{L}_\delta \end{bmatrix} \begin{bmatrix} \alpha \\ \delta \end{bmatrix} + \hat{l}. \quad (4.40)$$

Now we can already determine what the correct feedback should look like. If  $\delta > 0$  and  $\alpha = 0$ , the robot has to turn right. This means that for  $u < 0$ ,  $\hat{L}_\delta < 0$  is correct. If  $\alpha$  is also positive, the robot has to turn right even more. This means that also  $\hat{L}_\alpha < 0$  is correct. The  $\hat{l}$  can be used in that state to steer to the right even more, so this should also be negative. However, if  $\delta < 0$  the value of  $\hat{l}$  should be positive.

The model in (4.36) describes a smooth nonlinear function. This function is symmetric around the origin of the state space, which indicates that the optimal feedback function will go through the origin. So it is possible to use (4.39) as a local linear approximation of the optimal feedback function. When the robot is closely following the line, the value of  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$  will be very small. The extended LQRQL approach has to result in  $\hat{l} = 0$ .

For large values of  $\delta$ , the optimal feedback will move the robot in the direction of the line. This implies that  $\alpha$  is no longer very small, so that the nonlinearity in (4.36) will have an effect on the change of  $\delta$ . The value of  $\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{v})$  is no longer zero, so that the (4.39) will not be able to form a good local linear approximation of the optimal feedback function. The extended LQRQL approach can provide a better approximation for a large  $\delta$  and should result in  $\hat{l} \neq 0$ .

In order to compare the results of the different approaches, we need to have a criterion. The relative performance from the previous chapter cannot be used, because we do not know the optimal solution. We used the total costs over a fixed time interval

$$J = \sum_{k=0}^N r_k, \quad (4.41)$$

as criterion. The resulting feedback is obtained based on a train set, that was generated by starting at a certain state. To test the feedback we started the robot in the same state. As time interval we took the same number of time steps as used during the generation of the train set. By taking only a short time interval we made sure that we tested the feedback in the same local part of the state-action space where the train set was generated.

### 4.5.2 Experiment with a nonzero average $\boldsymbol{w}$ .

We first did an experiment with a nonzero average  $\boldsymbol{w}$ . We could do this by focusing on a part of the state space where  $\delta$  is large. Under these conditions the extended LQRQL approach should be able to perform better than the SI and standard LQRQL approach.

#### The setting

We took as settings:

- Sample time  $T = 0.35$  seconds.
- Number of time steps  $N = 57$ . So with  $T = 0.35$ , the robot drives for approximately 20 seconds.
- Gaussian exploration noise with  $\sigma_e = 0.1$ . A higher level of exploration would make it impossible to use similar settings on the real robot, because the tolerated rotation on the real robot is limited.
- Initial feedback  $L = \begin{bmatrix} -10^{-3} & -10^{-3} \end{bmatrix}$ . This feedback makes the robot move towards the line. On the other hand it is small enough, so that hardly any prior knowledge is included.
- Initial orientation  $\alpha = 0$ .

Because of the exploration we were not able to exactly determine the local part of the state space in which the train set is generated. The robot drives for approximately 20 seconds at a speed of 0.1 meters per second. Thus the robot traverses approximately 2 meters. As initial distance to the line we used  $\delta_0 = 1.5$  meters, for which the resulting  $\hat{l}$  should be negative. In the worst case the robot stops at  $\delta = -0.5$ , for which  $\hat{l}$  should be positive. On average most of the training samples were obtained in the part of the state space for which  $\hat{l}$  should be negative, so that we knew that the extended LQRQL had to result in a  $\hat{l} < 0$ . This means that the average  $\boldsymbol{w}$  was not zero. For the SI and standard LQRQL approach  $l = 0$ , so they should perform less than the extended LQRQL approach..

We generated one train set and used the SI, the standard LQRQL and the extended LQRQL approach. The resulting feedbacks of the three approaches were tested by starting the robot in the same initial state. For each test we computed the total costs according to (4.41). In Table 4.1 the resulting feedbacks and the total costs of the test runs are shown.

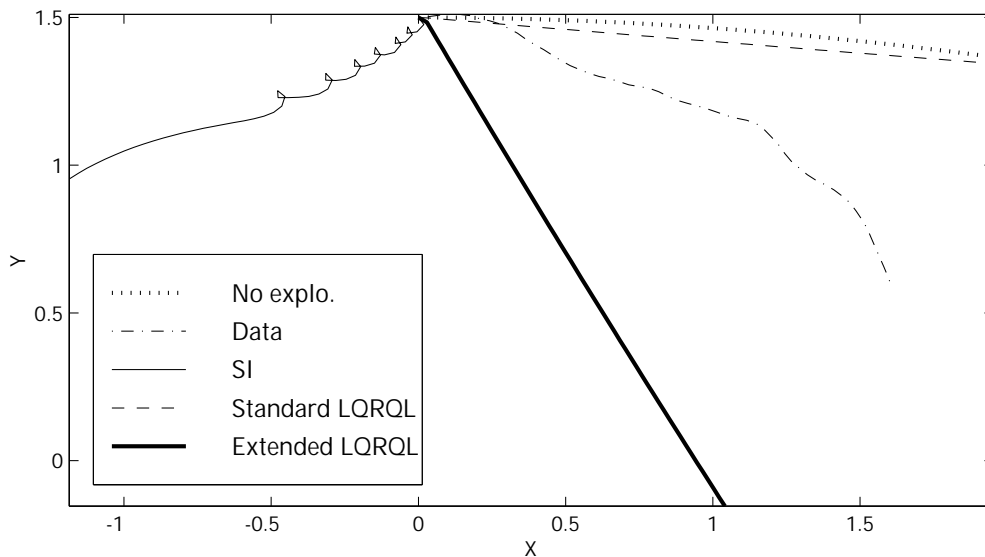
	$\hat{L}_\alpha$	$\hat{L}_\delta$	$\hat{l}$	J
SI	-0.2794	-0.7224	×	155.5828
Standard LQRQL	-0.5861	-0.0330	×	115.3245
Extended LQRQL	-0.5855	-0.0133	-0.5865	45.8943

**Table 4.1.** Results of the experiment with nonzero average  $\boldsymbol{w}$ .

The resulting value of  $\hat{l}$  is negative, just as we expected. The values of  $\hat{L}$  for both LQRQL approaches are almost the same, but the  $\hat{L}$  for the SI approach is different. The total costs of the SI approach is the highest and the extended LQRQL approach has the lowest total costs.

In figure 4.5 five trajectories are shown. The first shows that using the initial feedback without exploration will make the robot move slowly towards the line. The second trajectory shows that the generated train set, primarily depends on the exploration and not on the initial feedback. The other three trajectories are the test runs with the three resulting feedbacks.

The trajectory of the SI approach is the “curly” line in figure 4.5 that moves to the left. Because the value of  $\hat{L}_\delta$  is too large, the robot will rotate too much for large values of  $\delta$ . At the end of the trajectory the value of  $\delta$  is so small that the robot no longer rotates around its axis. The trajectory of the standard LQRQL approach resembles that of the initial feedback  $L$ , although  $L$  and  $\hat{L}$  are clearly different. The reason is that  $\hat{L}_\delta$  is too small, so the robot hardly rotates. Therefore the orientation  $\alpha$  will remain very small, so the higher value of  $\hat{L}_\alpha$  does not contribute to the control action. Although the extended LQRQL approach has approximately the same  $\hat{L}$ , because of  $\hat{l}$  it turns faster in the direction of the line. This explains the lower total costs. The nonzero value of  $\hat{l}$  suggests that the average  $\mathbf{w}$  is not zero. Since the extended approach can deal with this situation, it outperforms the other approaches.



**Figure 4.5.** The trajectories in the world. The line with “No Explo” is the trajectory when using the initial feedback  $L$ . For the line with “Data” the exploration is added to generate the train set. The other three lines are the trajectories of the test runs.

### 4.5.3 Experiments for different average $\mathbf{w}$

The previous experiment has shown that for a large value of  $\delta$  the extended LQRQL approach performs best. This is the case for a nonzero average  $\mathbf{w}$ . We also indicated that around the line the average value of  $\mathbf{w}$  is zero. Here the SI and standard LQRQL approach already have the correct value of  $l = 0$ . The extended approach has to estimate the correct  $\hat{l}$  which can result in an  $\hat{l} \neq 0$ . This suggests that the extended LQRQL approach will perform slightly less when the train set is generated near the line. We did some experiments to see how of the performance of the three approaches depends on the size of the average  $\mathbf{w}$ .

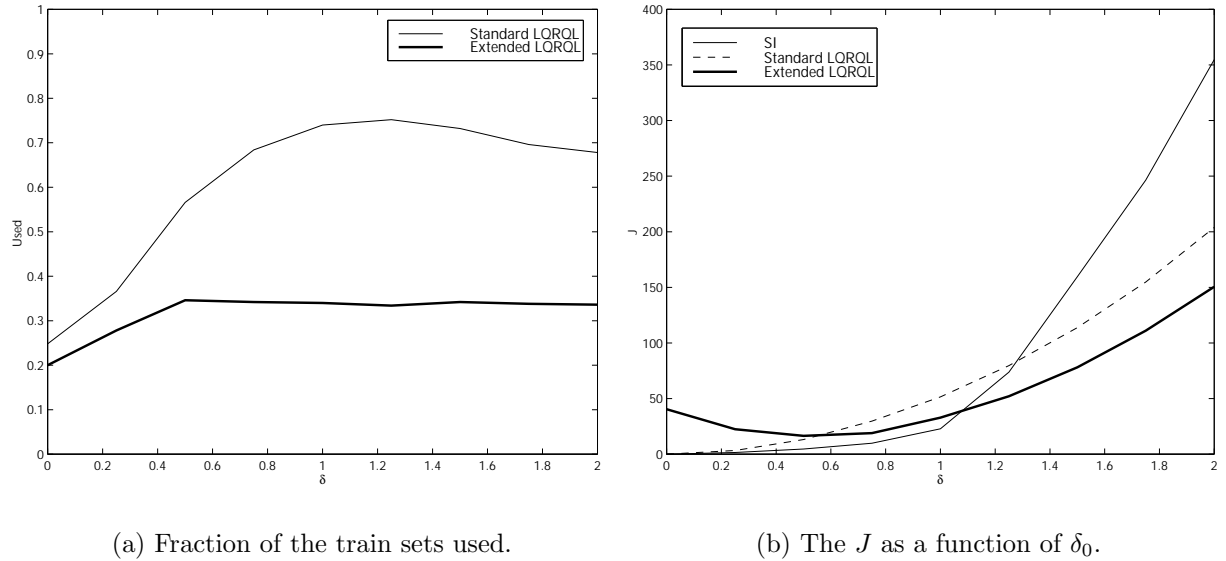
In the same way as in the previous experiment we used the initial distance  $\delta_0$  to determine the average  $\mathbf{w}$ . We varied the initial distance  $\delta_0$  from 0 to 2 in steps of  $\frac{1}{4}$ . The estimation results were based on train sets, generated with a random exploration noise. For this reason we generated for each  $\delta_0$  500 train sets, each with a different exploration noise sequence. The SI, standard LQRQL and extended LQRQL were applied to all train sets.

All resulting feedbacks can be tested, but we first looked at the reliability of the outcome. For all three approaches the resulting feedback function depends only on the train set. If the resulting feedback is not good then this is because the train set is not good enough. In chapter 3 we showed that this is the case when the amount of exploration is not enough. If sufficient exploration is used the estimated quadratic Q-function is positive definite. In that case the estimated matrix  $\hat{H}$  only has positive eigenvalues.

We used the same  $\sigma_e$  for all train sets. When we have a nonlinear system, we can see  $\mathbf{w}_k = \mathbf{w}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k)$  as system noise that does not have to be Gaussian or white. Since the amount of exploration must be higher than the amount of system noise, it can turn out that in some train set insufficient exploration was used. Another problem is that the contribution of the feedback function to the control action is very small when the train set is generated close to the line. The trajectory is mainly determined by the exploration. In these situations it can be very hard to evaluate the feedback function, so the estimated Q-function may not be correct. We know for sure that if  $\hat{H}$  has negative eigenvalues, the Q-function is not correct. For both LQRQL approaches we rejected the results with negative eigenvalues for  $\hat{H}$ .

The extended LQRQL approach also estimates  $\hat{l}$ , whose reliability does not depend on the eigenvalues of  $\hat{H}$ . If we look at figure 4.1 we see that a small change in the optimal feedback function may lead to a large change in the value of  $l$ . This means that a small difference in the train set may lead to large difference in the estimated  $\hat{l}$ . To be able to compare the results with the other two approaches, the situations for which  $\hat{l}$  were clearly wrong were also removed. For the extended LQRQL approach we rejected the results for which  $|\hat{l}| > 1$ . For these values of  $\hat{l}$ , the  $\hat{L}$  hardly contributed to the motion of the robot.

For the SI approach we do not have a criterion that indicates the reliability of the resulting feedback, so we used all train sets. Figure 4.6(a) shows the fractions of the 500 runs that were used for both LQRQL approaches. For the standard LQRQL approach about 75 percent was used. For the extended approach the percentage without negative



**Figure 4.6.** The results for different initial  $\delta$ .

eigenvalues for  $\hat{H}$  was already below 50 percent. When also the unreliable  $\hat{l}$  results were removed, about one third of the train sets were kept. This indicates that the extended LQRQL approach is the most sensitive to the particular train set.

The resulting feedback functions that were not rejected were tested. In figure 4.6(b) we see the average total costs  $J$  for different initial  $\delta$  for the three approaches. When figure 4.6(b) is plotted with error bars the total figure becomes unclear, therefore we omitted the errorbars. The maximal standard deviation was 40.8 for  $\delta = 2$  for the extended LQRQL approach. The figure clearly shows that the total costs increases as  $\delta$  increases, because  $\delta$  is part of the direct costs in (4.37). For  $\delta_0 = 0$  the SI and the standard LQRQL approach have  $J = 0$ . This is because the robot is already on the line and facing the right direction. So  $\mathbf{x} = 0$  and therefore also  $u = 0$ . Because  $\hat{l} \neq 0$  the total costs for the extended LQRQL approach is not zero.

For small values of  $\delta_0$ , the SI approach has the lowest costs, followed by the standard LQRQL approach. For high values of  $\delta_0$  the situation is reversed. The extended approach has the lowest total costs and the SI approach the highest. This is also what we observed in the preliminary experiment.

We can conclude that if the train set is generated close to the line, the average  $\mathbf{w}$  is almost zero. The SI and standard LQRQL approach perform best. Further away from the line, the average value of  $\mathbf{w}$  becomes larger. The total costs for the SI approach is much higher than that of the LQRQL approaches. The extended LQRQL approach has the lowest cost as expected.

## 4.6 Experiments on a Real Nonlinear System

### 4.6.1 Introduction

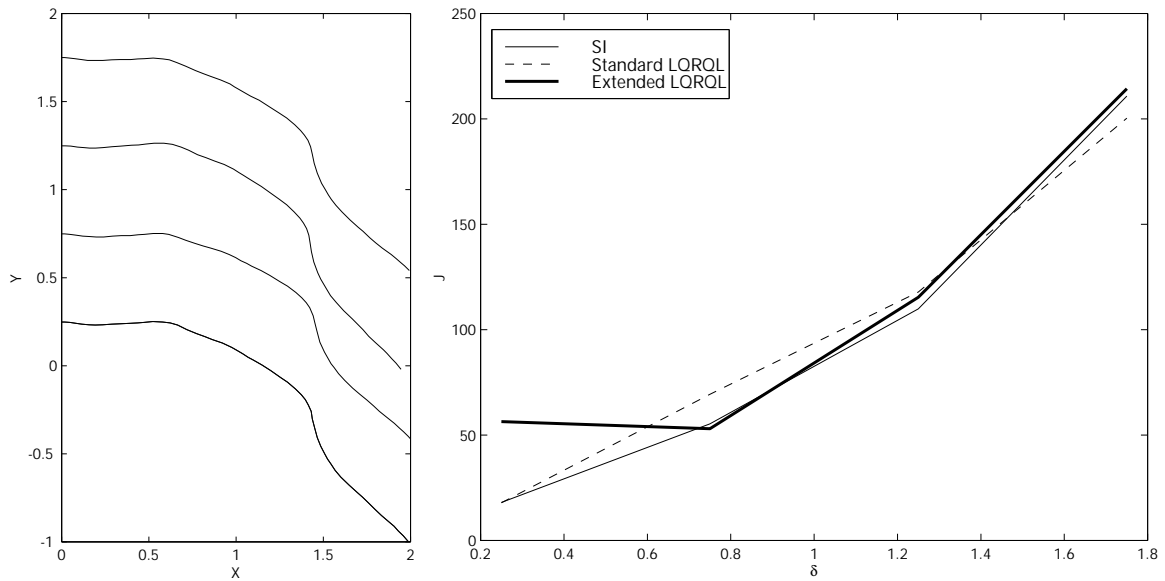
We did experiments with a real robot to see whether the simulation results also apply to real systems. The state value was derived from the odometry. This means that the robot keeps track of its position in the world by measuring the speed of both wheels. We then translated this into a value for  $\delta$  and  $\alpha$ .

There are some important differences with the simulated robot:

- The time steps do not have to be constant. State information is not directly available and is obtained via a communication protocol between the two processing boards in the robot. The duration of the communication varies a little. This introduces some noise, because at discrete time steps the state changes are influenced by the duration of the time step.
- The size of the control action that can be applied is bounded. Too high values for  $u$  may ruin the engine of the robot. When generating the train set it was possible that actions were tried that could not be tolerated. For safety we introduced the action bounds, such that  $u_{\min} = -0.18 \leq u \leq u_{\max} = 0.18$ . When a  $u$  outside this interval had to be applied to the robot, we replaced it by the value of the closest action bound. This value was also used for the train set. The consequence is that for high values of  $\sigma_e$ , the exploration noise is no longer Gaussian.
- The robot has a finite acceleration. This is a part of the dynamics of the system was not included in the model used for the simulations. This means that the trajectory during a time step is not exactly a circle. A consequence is that the robot might not respond so quickly on fast varying control action when exploring. So effectively the amount of exploration contributing to the movement of the robot is lower than the amount of exploration that is added to the control action.
- There is wheel spin. The effect of wheel spin is that the robot does not move according to the rotation of the wheels. This can mean that the real position of the robot does not change, while the wheels are rotating. Since we do not use the robot's real position, this does not affect us. But the odometry is based on measuring the speed of the wheels and they accelerate faster during wheel spin. This has the effect that similar control actions can lead to different state transitions.

### 4.6.2 The experiments

It was infeasible to generate the same amount of data sets as in the simulation experiments. We varied the initial  $\delta$  from 0.25 to 1.75 in steps of 0.5. We generated data for four different sequences of exploration noise, so in total we generated 16 data sets. For one exploration sequence the four data sets generated are shown in figure 4.7(a).



(a) The generation of four train sets, using the same exploration noise sequence.

(b) The average total costs  $J$  as a function of the initial  $\delta$ .

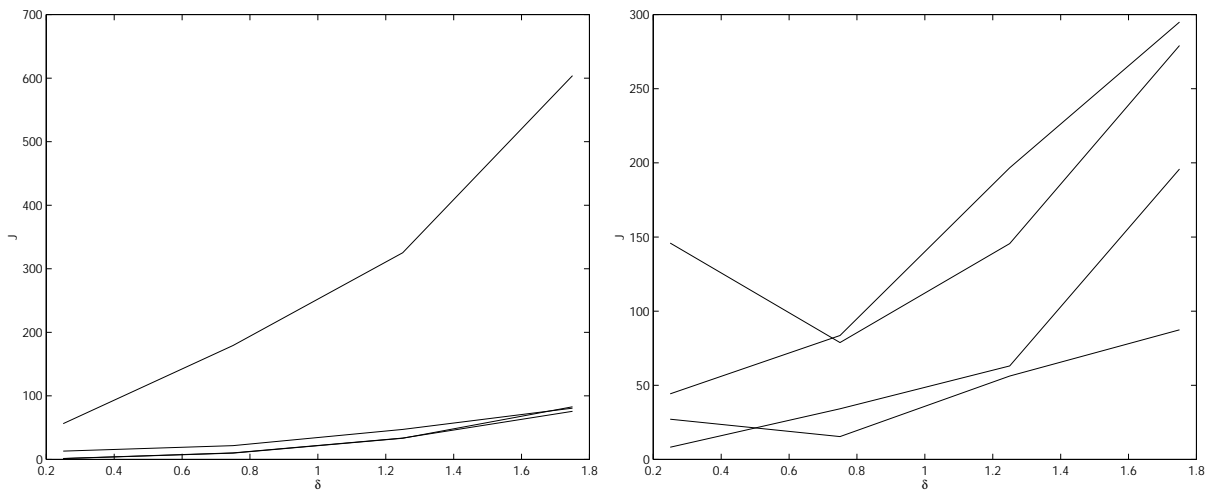
**Figure 4.7.** The experiment with the real robot.

In figure 4.7(b) we see the average total costs for each test of the three different methods. We did not remove any train set for the LQRQL approaches. The results in figure 4.7(b) show that on average all three approaches perform almost the same. Only the extended approach has higher total costs for  $\delta_0 = 0.25$ . The latter agrees with the simulation experiments. If we compare figure 4.7(b) with figure 4.6(b) we see that the performances of both LQRQL approaches are a little less in practice than in simulation. The SI approach seems to perform better in practice than in simulation.

In figure 4.8(a) we see, for all four exploration noise sequences, the total costs as a function of the initial  $\delta$  for the SI approach. For one sequence the total costs were very high (for  $\delta_0 = 1.75$  the total costs was more than 600.). This is because the resulting feedback function is wrong. It made the robot move away from the line. For the three other sequences we see that the total costs are very low. These low total costs are misleading. What happens in these situations is that the feedbacks found by the SI approach are much too high. This would lead to rotations as in figure 4.5, but the action bounds prevented the robot from rotating. Instead the action applied to the robot alternated between  $u_{\min}$  and  $u_{\max}$  for the first 15 to 20 time steps. After that the robot was very close to the line and it started following the line as it should. In these cases the low total costs were caused by the action bounds, which indicates that the SI approach did not find the optimal linear feedback!

The actions taken by the resulting feedbacks of both LQRQL approaches were always between  $u_{\min}$  and  $u_{\max}$ . In figure 4.8(b) we see for all four exploration noise sequences the total costs as a function of the initial  $\delta$  for the extended LQRQL approach. We noticed that the plot for the standard LQRQL approach looks quite similar. We see that for some exploration sequences the costs are quite low while for some the costs are higher. This indicates that the performance depends on the particular sequence of exploration noise. For one sequence the total costs are as low as for the three sequences of the SI approach. Only the extended approach did not exploit the action bounds. This indicates that the extended LQRQL approach optimized the local linear feedback function for the real nonlinear system.

In the simulation experiments we rejected train sets that resulted in negative eigenvalues for  $\hat{H}$  because we considered them unreliable. We did not do this for the experiments on the real robot. For the standard LQRQL approach 9 of the 16 train sets resulted in a positive definite  $\hat{H}$  and for the extended LQRQL approach only 5. The total costs for these feedbacks were not always the lowest, because in some cases an “unreliable” feedback performed better. This was caused by the acceleration of the robot. The reliable feedbacks made the robot rotate faster in the direction to the line than the unreliable feedbacks. But due to the acceleration, the robot did not always stop rotating fast enough. Then the robot was facing the wrong direction and had to rotate back. Many unreliable feedbacks turn slower towards the line and are therefore less affected by the acceleration. This implies that the reliability indication based on the eigenvalues of  $\hat{H}$  is not appropriate for the real robot if the acceleration is ignored.



(a)  $J$  for SI as function of  $\delta_0$  for all four exploration noise sequences.

(b)  $J$  for extended LQRQL as function of  $\delta_0$  for all four exploration noise sequences.

**Figure 4.8.** All 16 performances for SI and extended LQRQL.

In summary, the experiments have shown that the SI approach does not optimize a local linear feedback correctly. Instead it either finds a wrong feedback or it finds feedbacks that are too large, so that most control actions are limited by the action bound. The performances of the standard and extended LQRQL approach are similar to those of the simulation experiments. So we can conclude that they correctly optimize a local linear feedback.

## 4.7 Discussion

We started this chapter with a presentation of a number of methods for the control of nonlinear systems. Some of these methods were based on local linear approximations of the system. We were interested whether we could use Q-Learning to find a feedback which is locally optimal. We showed that the standard LQRQL approach as presented in chapter 3 would not lead to an optimal linear feedback, and introduced the *extended* LQRQL approach. In this approach we do not estimate the parameters of a global quadratic function through the origin, but we use the data to estimate the parameters of a more general quadratic function. The consequence for the feedback function is that an extra constant was introduced. In this way the resulting feedback is no longer restricted to a linear function through the origin.

We tested the extended LQRQL approach on a nonlinear system in simulation and on a real nonlinear system. We compared it with the SI and standard LQRQL approach from chapter 3. The results indicate that if we explore in a restricted part of the state space, by sampling for a few time steps, the extended approach performs much better. This means that if a feedback function is based on local linear approximations, the extended approach has to be used. The standard LQRQL should only be used if it is known that the optimal feedback function goes through the origin.

It is possible to use multiple linear models to construct one global nonlinear feedback function. In [44] bump-trees were used to form a locally weighted feedback function. In [17] local linear feedbacks were used for different partitions of the state space of a nonlinear system. The result is equivalent to Gain Scheduling as described in chapter 2. In both cases the resulting local linear feedback functions were obtained by the standard LQRQL approach.

However, for standard LQRQL the local linear models are not only based on the train set but also on the position of the partition with respect to the origin. This means the models are not completely local. A consequence is that the linear feedbacks for partitions far away from the origin will become smaller. This is because the feedback has to go through the origin. The consequence is that it is unlikely that linear models far away from the origin are optimal. This implies that if we want to form a nonlinear feedback based on local linear feedbacks, we have to use the extended approach. The extended LQRQL approach is able to estimate the appropriate set point for each linear model.

## 4.8 Conclusions

Many control design techniques for nonlinear systems are based on local linear approximations. We studied the use of LQRQL for obtaining a local linear approximation of a nonlinear feedback function. A nonlinear system can be regarded as a linear system with an additional nonlinear correction. If in a local part of the state space the average correction is not zero, the SI and standard LQRQL approach from chapter 3 will approximate the wrong function. We introduced the extended LQRQL approach, that will result in a linear feedback plus an additional constant. The experiments on a nonlinear system have shown that if the additional constant is not zero, the extended approach will perform better.

# Chapter 5

## Neural Q-Learning using LQRQL

### 5.1 Introduction

The LQRQL approach was derived for linear systems with quadratic costs. Although this approach can also be applied to nonlinear systems or other cost functions, the resulting feedback will always be linear. This is a consequence of the restriction that the Q-function is quadratic. The use of a quadratic Q-function was motivated by the possibility to use a linear least squares estimation to obtain the parameters of the Q-function. Then the parameters of the linear feedback follow directly from the parameters of the Q-function.

If the system is nonlinear, it is very likely that the Q-function is not quadratic. In such situations it is possible to use a general approximator, like a feed forward network, to approximate the Q-function. The feedback can be obtained by finding for all states the control action that minimizes the feedback function. We will show that this can have the effect that the feedback function is not a continuous function and we will explain why it is desirable to have a continuous feedback function. In order to have a continuous feedback function, another network can be used to represent the feedback function. This is the actor/critic configuration as described in section 2.3. This has the drawback that two networks have to be trained, where the training of one network depends on the other. This makes the training very hard.

The choice is either to have a discontinuous feedback or to use two networks. In this chapter we propose a method in which we only use one network and still have a continuous feedback function. The idea is to combine LQRQL with a carefully chosen network and we called this Neural Q-Learning. The feedback can be derived in the same way as in LQRQL.

The simulation and real experiments were based on the same nonlinear system as in chapter 4. We applied the Neural Q-Learning method and compared the result with Gain Scheduling based on the extended LQRQL approach. The resulting nonlinear feedback functions have to be globally valid. So instead of focusing on a local part of the state space, we looked at larger parts of the state space.

## 5.2 Neural Nonlinear Q-functions

Instead of a quadratic Q-function, a general approximator like a feed forward neural network can be used. It describes the function

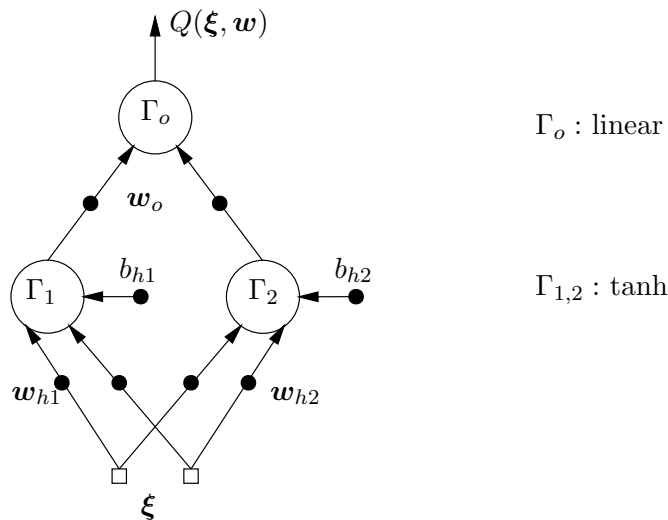
$$Q(\boldsymbol{\xi}, \boldsymbol{w}) = \Gamma_o(\boldsymbol{w}_o^T \boldsymbol{\Gamma}_h(W\boldsymbol{\xi} + \boldsymbol{b}_h) + b_o), \quad (5.1)$$

with  $\Gamma_o$  and  $\Gamma_h$  the activation functions of the units in the output and hidden layer. The rows of matrix  $W$  contain the weight vectors  $\boldsymbol{w}_{h,i}$  of the hidden units and the vector  $\boldsymbol{b}_h$  contains the corresponding biases. The weights of the output unit are given by  $\boldsymbol{w}_o$  and the bias by  $b_o$ . The network has only one output  $Q(\boldsymbol{\xi}, \boldsymbol{w})$ , where the vector  $\boldsymbol{w}$  indicates all weights of the network. The  $\boldsymbol{\xi}$  represents the input of the Q-function.

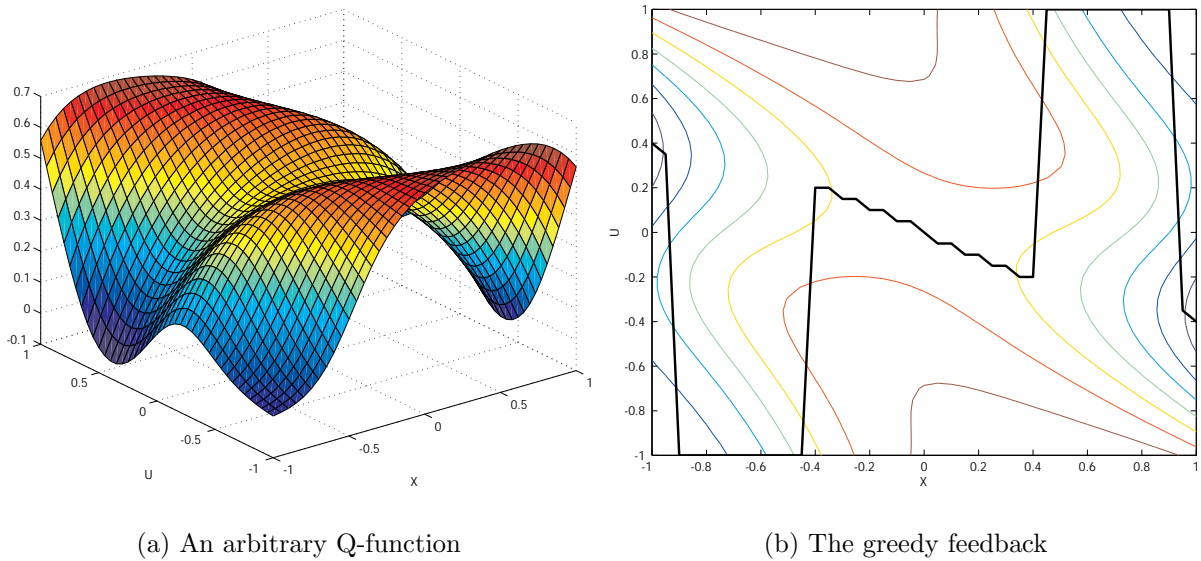
If a network is used to represent a Q-function we might obtain a typical function as in figure 5.2(a). This is drawn for a scalar state and control vector, and the weights of the network are picked randomly. It is clear that this Q-function is a smooth continuous function. Given this Q-function the feedback has to be determined.

According to (2.15) and (3.15) the greedy feedback is computed by taking the minimum value of the Q-function for each state. In figure 5.2(b) the top view of the Q-function is shown with lines indicating similar Q-values. In this plot also the greedy feedback function is shown that is computed by taking for each  $x$  the value of  $u$  for which the Q-value is minimal. Using the greedy feedback from the network has two drawbacks:

- The computation of the greedy action involves finding the extremum of a nonlinear function. In general this is not a trivial task.
- Even when the Q-function is smooth as in figure 5.2(a), it is still possible that the greedy feedback function (shown in figure 5.2(b)) is not a continuous function. This



**Figure 5.1.** The network with two hidden units



**Figure 5.2.** The Q-function is formed by a feed forward network. In (b) we see the top view of the Q-function with lines indicating the height. The bold line indicates the greedy feedback.

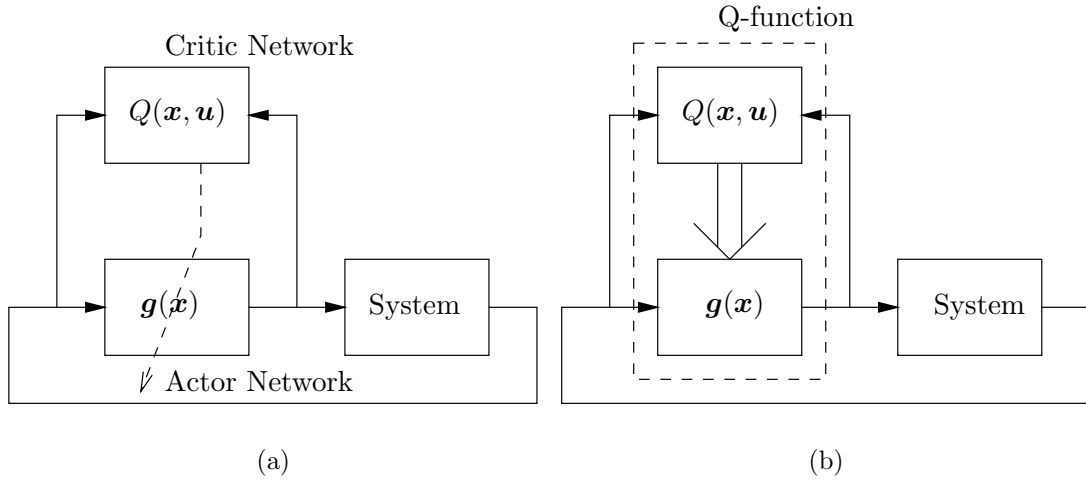
can cause problems for real applications when there is noise in the system. Near the discontinuities a little noise can have a high influence on the control action, so the behavior of the system becomes very unpredictable.

The second drawback can be overcome by introducing a Dynamic Output Element [60]. The static feedback is followed by a low pass filter, removing the effects of noise at discontinuities of the feedback function. In this way the control action is not only based on the current state value, but also on the previous control actions. This approach is not a solution to the first drawback.

One approach that removes both drawbacks is the Actor/Critic approach described in chapter 2. A second function approximator is introduced to represent the feedback function. If a continuous differential function is used as actor, the second drawback is overcome. By training the actor based on the critic also the first drawback is removed.

The main difference between the actor/critic configuration and Q-Learning is shown in figure 5.3 and is the way in which the feedback is derived from the Q-function. In the actor/critic configuration the actor is trained based on the critic. In the Q-Learning configuration the Q-function implicates the feedback, so that the Q-function and feedback are represented by *one and the same* network.

The two networks in the actor/critic configuration have to be trained, and this is the major problem of this configuration. Training two networks means that training parameters and initial settings have to be selected for two networks. It is very hard to determine beforehand the appropriate setting for the training of the networks. Since the settings influence the result, the interpretation of the results is very hard.



**Figure 5.3.** The actor/critic configuration and Q-learning. The dashed arrow indicates the training of the actor based on the critic. The implication arrow indicates that the feedback function directly follows from the Q-function.

In the LQRQL approach the greedy feedback is computed by setting the derivative of the Q-function to the control action to zero. So for LQRQL the quadratic Q-function implicates the linear feedback function, and the parameters of the feedback function can be expressed as functions of the parameters of the Q-function. This is a property we want to keep for the neural Q-function.

We will propose a method that uses LQRQL to keep this property for a neural Q-function. In this method the feedback function can be expressed using the weights of the neural Q-function. No second network is required. The method will also guarantee that there are no discontinuities in the feedback function.

### 5.3 Neural LQRQL

Neural LQRQL<sup>1</sup> is based on the standard LQRQL described in chapter 3, but then with a feed forward network to represent the Q-function. The standard LQRQL approach is based on three steps. First the least squares estimation is applied to the train set, resulting in an estimation  $\theta$ . Then the parameters of the quadratic Q-function  $\hat{H}$  are derived from the estimated  $\theta$ . Finally the value of linear feedback  $\hat{L}$  is computed based on  $\hat{H}$  using (3.16). This indicates that if it is possible to derive  $\hat{\theta}$  from the neural Q-function, the feedback can be computed from  $\hat{\theta}$  analogue to the LQRQL approach.

In LQRQL the quadratic Q-function was represented as a linear multiplication of the quadratic combinations of the state and action with the estimated parameters. This was introduced to make it possible to use a linear least squares estimation for estimating  $\theta$

<sup>1</sup>An almost similar approach in [33] is called Pseudo-Parametric Q-Learning.

based on the measurements. In section 3.2.4 the Q-function is written as

$$Q^L(\mathbf{x}_k, \mathbf{u}_k) = \text{vec}^{\sphericalangle}(\boldsymbol{\phi}_k \boldsymbol{\phi}_k^{\text{T}})^{\text{T}} \text{vec}^{\sphericalangle}(H^L) = \boldsymbol{\Omega}_k^{\text{T}} \boldsymbol{\theta}. \quad (5.2)$$

Here  $H^L$  represents the parameters of the quadratic Q-function and  $\boldsymbol{\phi}_k$  the vector with the state and control action. This can be regarded as just a multiplication of two vectors  $\boldsymbol{\Omega}_k$  and  $\boldsymbol{\theta}$ . We see that writing the quadratic Q-function as a linear function requires that the input vector is “quadratic”. Instead of having a quadratic function with input  $\boldsymbol{\phi}$ , we have a linear function with input  $\boldsymbol{\Omega}$ . Input  $\boldsymbol{\Omega}$  contains all quadratic combinations of elements of  $\boldsymbol{\phi}$  and vector  $\boldsymbol{\theta}$  contains the corresponding values from  $H^L$ .

The representation of the Q-function in (5.2) can also be viewed as a one layer feed-forward network with a linear transfer function and input  $\boldsymbol{\Omega}_k$ . We can extend this representation by writing this similar to (5.1):<sup>2</sup>

$$Q(\mathbf{x}_k, \mathbf{u}_k) = \Gamma_o(\mathbf{w}_o^{\text{T}} \boldsymbol{\Gamma}_h(W \boldsymbol{\Omega}_k^{\text{T}}) + \mathbf{b}_h) + b_o. \quad (5.3)$$

If we take  $\Gamma_o$  and  $\boldsymbol{\Gamma}_h$  as linear functions and the biases  $\mathbf{b}_h$  and  $b_o$  all zero, then there exist values for  $W$  and  $\mathbf{w}_o$  such that (5.3) equals (5.2). Note that these weights are not unique, because the number of weights is higher than the number of parameters of (5.3). Since we want to stay as close as possible to the original LQRQL approach, we will only consider activation functions that resemble linear functions.

The neural representation is introduced to deal with non quadratic Q-functions. The network in (5.3) with linear transfer functions can not represent these functions, therefore we have to use nonlinear transfer functions. Let the output of hidden unit  $i$  is given by:

$$\Gamma_{h,i}(\mathbf{w}_{h,i}^{\text{T}} \boldsymbol{\Omega}_k^{\text{T}} + \mathbf{b}_{h,i}) = \tanh(\mathbf{w}_{h,i}^{\text{T}} \boldsymbol{\Omega}_k^{\text{T}} + \mathbf{b}_{h,i}). \quad (5.4)$$

The hyperbolic tangent is nonlinear, but in the origin it is zero and its derivative is one. So for small values of  $\mathbf{w}_{h,i}$  (5.4) still resembles a unit with a linear transfer function. Only when the weights of the hidden units become large, the Q-function will no longer be a quadratic function.

### 5.3.1 Deriving the feedback function

Given the Q-function and its input, the values of  $\hat{\boldsymbol{\theta}}$  should be obtained. In the case of the standard LQRQL these are just the parameters that are estimated, so they are immediately available. For the neural Q-function this is different. The weights are the parameters that are estimated based on the train set, so the value of  $\hat{\boldsymbol{\theta}}$  should be derived from the weights.

Given  $Q^L(\boldsymbol{\Omega}_k)$  from (5.2), the parameters  $\boldsymbol{\theta}$  can also be obtained by computing the derivative of this function to the input  $\boldsymbol{\Omega}$ . This means that parameter  $\theta_i$  can be computed according to:

$$\theta_i = \frac{\partial Q^L(\boldsymbol{\Omega}_k)}{\partial \Omega_i}. \quad (5.5)$$

---

<sup>2</sup>This  $Q$  function still depends on the feedback function used to generate the data, which can also be a nonlinear function. Therefore we will omit the superscript  $L$ .

This shows that the  $\theta_i$  does not depend on  $\mathbf{\Omega}_k$ , so the values of  $\theta_i$  do not depend on the state and control action. When this is computed for all parameters of  $\boldsymbol{\theta}$ , then  $H^L$  can be derived and therefore  $\tilde{L}$  can be computed.

In the same way  $\hat{\theta}_i(\mathbf{\Omega})^3$  can be computed for the neural Q-function in (5.3):

$$\hat{\theta}_i(\mathbf{\Omega}_k) = \frac{\partial Q(\mathbf{\Omega}_k)}{\partial \Omega_i} = \sum_{j=1}^{n_h} w_{o,j} w_{h,i,j} (1 - \tanh^2(\mathbf{w}_{h,i}^T \mathbf{\Omega}_k + b_{h,i})) \quad (5.6)$$

where  $w_{h,i,j}$  represents the weight from input  $\Omega_i$  to unit  $j$  and  $n_h$  indicates the number of hidden units.

When  $\hat{\boldsymbol{\theta}}(\mathbf{\Omega})_k$  is available, it can be rearranged into  $\hat{H}(\mathbf{\Omega})$  so that  $L(\mathbf{\Omega})$  can be computed. The control action can be computed according to  $\mathbf{u}_k = L(\mathbf{\Omega}_k)\mathbf{x}_k$ . This is still a linear multiplication of the state with  $L(\mathbf{\Omega}_k)$ , but it is a nonlinear feedback function because  $\mathbf{\Omega}_k$  contains  $\mathbf{x}_k$ . The problem is that it also contains  $\mathbf{u}_k$ , which is the control action that has to be computed.

In order to solve this problem (5.6) can be split into two parts:

$$\hat{\theta}_i = \underbrace{\sum_{j=1}^{n_h} w_{o,j} w_{h,i,j}}_{\text{linear}} - \underbrace{\sum_{j=1}^{n_h} w_{o,j} w_{h,i,j} \tanh^2(\mathbf{w}_{h,i}^T \mathbf{\Omega}_k + b_{h,i})}_{\text{nonlinear}}. \quad (5.7)$$

The first part is indicated with “linear”, because this corresponds to the  $\hat{\theta}_i$  that would be computed when all hidden units are linear. The resulting feedback function derived from this network would also be linear. The second part is indicated with “nonlinear” because this is the effect of the nonlinearities introduced by the hyperbolic tangent transfer functions.

A linear feedback can be obtained from the network by just ignoring the hyperbolic tangent functions. Define  $\tilde{\theta}_i$  by:

$$\tilde{\theta}_i = \sum_{j=1}^{n_h} w_{o,j} w_{h,i,j}. \quad (5.8)$$

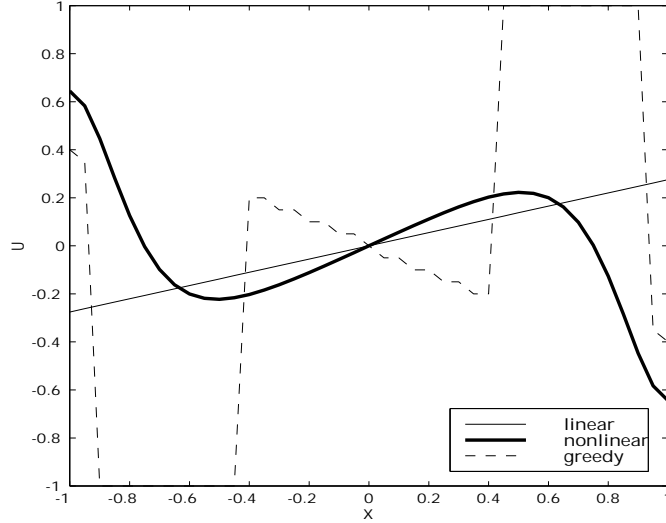
This leads to a vector  $\tilde{\boldsymbol{\theta}}$ . Analog to  $\hat{\boldsymbol{\theta}}$  we can rearrange  $\tilde{\boldsymbol{\theta}}$  to form a quadratic function with parameters  $\tilde{H}$ . From  $\tilde{H}$  we can derive a linear feedback  $\tilde{L}$ .

The feedback  $\tilde{L}$  can be used to compute the control action  $\tilde{\mathbf{u}} = \tilde{L}\mathbf{x}$ . This control action can be used to control the system, but it can also be used to obtain the vector  $\tilde{\mathbf{\Omega}}$ . The vector  $\tilde{\mathbf{\Omega}}$  represents the quadratic combination of the state  $\mathbf{x}$  with the control action  $\tilde{\mathbf{u}}$ . This can be used to obtain the nonlinear feedback

$$\mathbf{u}_k = (H_{uu}(\tilde{\mathbf{\Omega}}_k))^{-1} H_{ux}(\tilde{\mathbf{\Omega}}_k) \mathbf{x}_k = L(\tilde{\mathbf{\Omega}}_k) \mathbf{x}_k. \quad (5.9)$$

Here the linear feedback  $L(\tilde{\mathbf{\Omega}}_k)$  is a function of  $\mathbf{x}_k$ , so that the resulting feedback function is nonlinear. If we compute this feedback function for the Q-function in figure 5.2(a) then this results in the feedback function shown in figure 5.4.

<sup>3</sup>Strictly speaking, the  $\hat{\theta}_i(\mathbf{\Omega})$  is not an estimation. For consistency with the previous chapters we will keep indicating it with a hat.



**Figure 5.4.** Nonlinear feedback example. The feed forward network forms the same Q-function as in figure 5.2(a). This figure shows the resulting linear feedback  $\tilde{L}$  and the nonlinear feedback function according to (5.9). For comparison also the greedy feedback from figure 5.2(b) is plotted.

### 5.3.2 Discussion

If we want to interpret the resulting feedback function, we have to look at (5.6). In case  $\mathbf{w}_{h,i}^\top \boldsymbol{\Omega}_k + b_{h,i}$  is large for a hidden unit  $i$ , the  $(1 - \tanh^2(\mathbf{w}_{h,i}^\top \boldsymbol{\Omega}_k + b_{h,i}))$  will be very small. The contribution of this hidden unit to  $\hat{\theta}_i$  will be very small. In case  $\mathbf{w}_{h,i}^\top \boldsymbol{\Omega}_k + b_{h,i}$  is zero for a hidden unit  $i$ , the  $\tanh(\mathbf{w}_{h,i}^\top \boldsymbol{\Omega}_k + b_{h,i})$  will be very small. The hyperbolic tangent can be ignored and the contribution of this hidden unit is the same as for the computation of the linear feedback  $\tilde{L}$ . We therefore can interpret the feedback function as a locally weighted combination of linear feedback functions.

Each hidden unit will lead to a linear feedback function and the value of the state  $\mathbf{x}$  determines how much the feedback function of each state contributes to the total feedback. When we compute  $\tilde{L}$  all hidden units are weighted equally. This results in a linear feedback function that is globally valid, because it no longer depends on the state value. When the neural approach is applied to the LQR task, then the resulting feedback should not depend on the state value. The weights of the hidden units should be very small, so that the feedback function should become linear.

The linear feedback  $\tilde{L}$  does not have to be the same as the result  $\hat{L}_{QL}$  of LQRQL, when applied to the same data set. In standard LQRQL the parameters of a quadratic Q-function are estimated. When the Q-function to approximate is not quadratic, then a small part of the train set can have a large influence on the estimated parameters. This is the case for training samples obtained in parts of the state space where the Q-function deviates a lot from a quadratic function. If (5.3) is used to approximate the Q-function, the influence of these points is much less. One hidden unit can be used to get a better approximation of the true Q-function for that part of the state space. Since  $\tilde{L}$  is based on

the average linear feedback formed by the hidden units, the contribution of the training samples on the resulting linear feedback  $\tilde{L}$  is much smaller. This means that if the true Q-function is not quadratic, the linear feedback derived from the network is more reliable than the result of the standard LQRQL approach.

The linear feedback  $\tilde{L}$  can always be derived from the network. This does not hold for the nonlinear  $L(\tilde{\Omega}_k)$ , because it depends on the weighting of the linear feedbacks based on the state value. It might be possible that for some state values, the contribution of *all* hidden units to  $\tilde{\theta}$  are zero. Then no feedback can be computed. This can only occur when all weights of the hidden units are large. This means that it is important to prevent the weights from becoming too large. One way of doing this is by incorporating a form of regularization in the learning rule. Another way is to scale down the reinforcements  $r$ , so that the Q-function to approximate becomes smoother. The easiest way is to make sure that the number of hidden units is not too large, so that it becomes very unlikely that the weights become too large.

## 5.4 Training the Network

The weights of the network in (5.3) can be found by training the network according to one of the methods described in section 2.3. If the method that minimizes the quadratic temporal difference error (2.24) using (2.25) and (2.26) with a discount factor  $\gamma = 1$  is used, then the training is based on minimizing the same criterion as (3.25). The only difference is that the network starts with some random initial weights that are incrementally updated at each iteration step. The least squares estimation gives the global minimum of the error at once. A consequence of this difference is that the weights of the network always have a value, while the least squares estimation gives no solution in case of a singularity. On the other hand, the training of the network can fail to find the global minimum of the error.

Because the network approximates a Q-function (2.24) is written using  $Q(\boldsymbol{\Omega}, \mathbf{w})$ :<sup>4</sup>

$$E = \frac{1}{2} \sum_{k=0}^{N-1} (r_k + Q(\boldsymbol{\Omega}_{k+1}, \mathbf{w}) - Q(\boldsymbol{\Omega}_k, \mathbf{w}))^2. \quad (5.10)$$

and (2.26) becomes

$$\Delta \mathbf{w}_k = (r_k + Q(\boldsymbol{\Omega}_{k+1}, \mathbf{w}) - Q(\boldsymbol{\Omega}_k, \mathbf{w}))(\nabla_{\mathbf{w}} Q(\boldsymbol{\Omega}_{k+1}, \mathbf{w}) - \nabla_{\mathbf{w}} Q(\boldsymbol{\Omega}_k, \mathbf{w})). \quad (5.11)$$

Before the network can be trained, the initial weights have to be selected. If the weights of the hidden units are chosen very small, the feedback function derived from this initial network is linear. If the true Q-function is quadratic, the weights change such that the feedback remains linear. Only when it is necessary, when the true Q-function is not quadratic, the weights of the hidden units become larger. Then the resulting feedback will be linear. In this way it is prevented that the resulting feedback becomes very nonlinear, while the simpler linear feedback is also possible.

<sup>4</sup>When the discount factor  $\gamma = 1$ .

There are some differences between the estimation of the parameters of a quadratic function and the training of the network. The important differences are:

- **No singularity**

For too low exploration no feedback can be computed for the standard approach. The neural approach already starts with an initialized network and the network only changes its parameters. This means that there is no singularity for too low exploration. Singularity can only happen when the weights of the hidden units are too large so that  $\tilde{\theta}$  is almost zero.

- **Different results for low exploration**

The training of the network is such that the temporal difference is minimized. Even if the global minimum is found this does not necessary result in an improved feedback function. If insufficient exploration is used then LQRQL results in the feedback used to generate the train set. It is very unlikely that the feedback derived from the network will be the same as the feedback used to generate the train set. Even when the same train set is used, the resulting feedbacks will not be the same when the networks are initialized differently.

- **Improvement for high exploration**

Sufficient exploration is achieved, if the exploration contributes to the temporal difference error. Only then, the minimization of the temporal difference error will lead to an improvement of the feedback function. The exploration will make the temporal difference error much higher. This means that a network, trained on a set with insufficient exploration, will have a lower error than one trained on a set with sufficient exploration. For a train set with sufficient data, different networks initialized with small weights will eventually result in similar feedback functions.

## 5.5 Simulation Experiments with a Nonlinear System

In chapter 4 we used a mobile robot to experiment with the different approaches. We used the same system to test the Neural Q-Learning approach. The main difference with chapter 4 is that the resulting feedback function should be globally valid. This means that we test the feedback function in a larger part of the state space. In order to compare the results with a different nonlinear feedback, we also used Gain Scheduling based on the Extended Q-Learning approach.

### 5.5.1 Introduction

The system was identical to the one in chapter 4, we only have to specify the settings for the two approaches. To be able to compare the results both approaches used the same train sets. This means that the train set was generated with one linear feedback  $L = \begin{bmatrix} 10^{-3} & 10^{-3} \end{bmatrix}$ .

## Gain Scheduling

For the Gain Scheduling approach we had to partition the state space in separate partitions. For each partition one local linear feedback was obtained by using extended LQRQL. For each estimation a train set was selected consisting of all training samples in that particular partition. This means that we had to make sure that for each partition sufficient training samples were available to obtain a local feedback. We did this by generating more train sets by starting the system from different initial states.

In chapter 4 we observed that different feedbacks were found when train sets were generated with a different initial  $\delta$ . This implies that the feedback should be different for different values of  $\delta$ . We therefore divide the state space into three partitions according to:

- Negative partition:  $-\infty < \delta < -1$
- Middle partition:  $-1 < \delta < 1$
- Positive partition:  $1 < \delta < \infty$

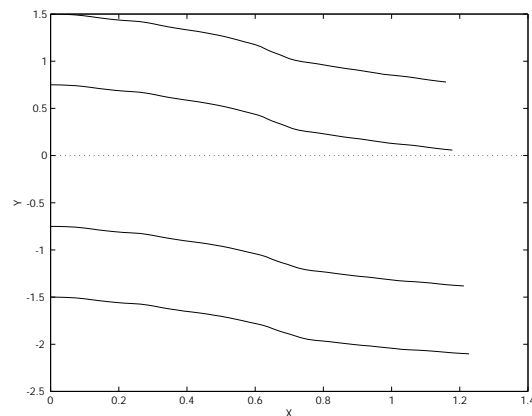
The resulting feedback function consists of three local linear functions. At the border of the partitions these functions can be different, so that the feedback function is not continuous. It is possible to make this a continuous differential function by smoothing it near the borders of the partitions. We did not do this because we wanted to see the local linear feedbacks as clearly as possible for the comparison with the Neural Q-Learning approach.

## Neural Q-Learning

For the Neural Q-Learning approach, all train sets used for Gain Scheduling were combined into one train set. This train set was used to train the network, by minimizing the quadratic temporal difference error using (5.10). Because the result also depends on the initial weights, we trained the network for ten different initial weights. Then we used the network with the lowest quadratic temporal difference error. To prevent over fitting it is possible to split the train set in two and use only one set for training and the other for testing. We did not do this because we wanted the resulting feedback to be based on the same train set as the Gain Scheduling approach. Instead we made sure that we did not have too many hidden units.

We chose the following settings for the network:

- Number of hidden units: 3.  
This was to prevent over fitting and to keep the number of weights close to the number of parameters of the Gain Scheduling approach.
- The value of the initial output weights: 0.
- Values of initial weights and biases of hidden units: Random between  $-10^{-4}$  and  $10^{-4}$ .



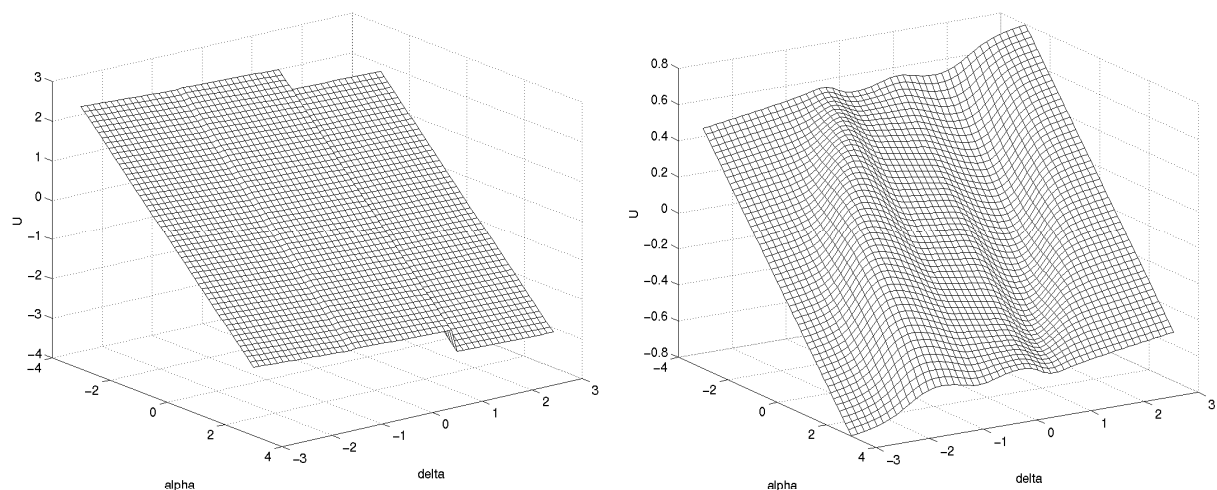
**Figure 5.5.** The trajectories while generating the train set.

These weights are taken small so that the initial resulting feedbacks are linear and perhaps become nonlinear during training.

### 5.5.2 The experimental procedure

The purpose of this experiment is to point out the major differences with the experiments in chapter 4. Different train sets are generated by starting the system for different initial  $\delta$ . For sequence of exploration the train sets are shown in figure 5.5.

All train sets are combined to form one train set that is used to obtain the two global



(a) Gain Scheduling

(b) Neural Q-Learning

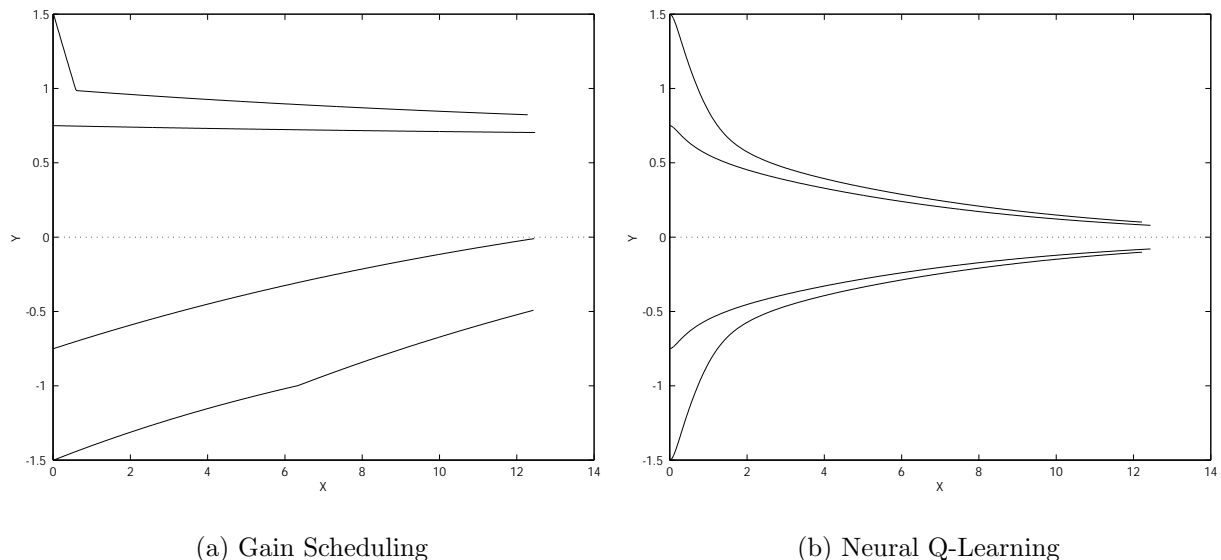
**Figure 5.6.** The control action as function of the state

nonlinear feedback functions. The resulting feedback functions of the train sets in figure 5.5 are shown in figure 5.6. In figure 5.6(a) the Gain Scheduling result is shown. The feedback function consists of three local linear feedbacks, but in this case two linear feedbacks are quite similar. The third feedback is different and the “jump” between the boundary of the two partitions is very clear. In figure 5.6(b) the resulting feedback based on Neural Q-Learning is shown. This looks like one linear feedback with a smooth nonlinear correction.

The feedback functions were tested by starting the robot for different initial  $\delta$ . In figure 5.7(a) four trajectories are shown for the Gain Scheduling result. The trajectory that starts in  $\delta = 1.5$  clearly shows a jump when it crosses the boundary between the partitions of the state space. After the jump the trajectory moves similar to the other trajectories in that partition towards the line.

In figure 5.7(b) the resulting trajectories for Neural Q-Learning are shown. It is clear that there are no jumps in the trajectories. Also it can be noticed that for large initial distances to the line,  $\delta = -1.5$  and  $\delta = 1.5$ , the robot initially moves faster towards the line. This is a consequence of the nonlinear correction.

The main difference with the experiments in chapter 4 is that now two global nonlinear feedbacks are used. This means that the tests from different initial  $\delta$  are performed with the same feedback function. In chapter 4, each initial  $\delta$  was tested with the corresponding local linear feedback.



**Figure 5.7.** The trajectories

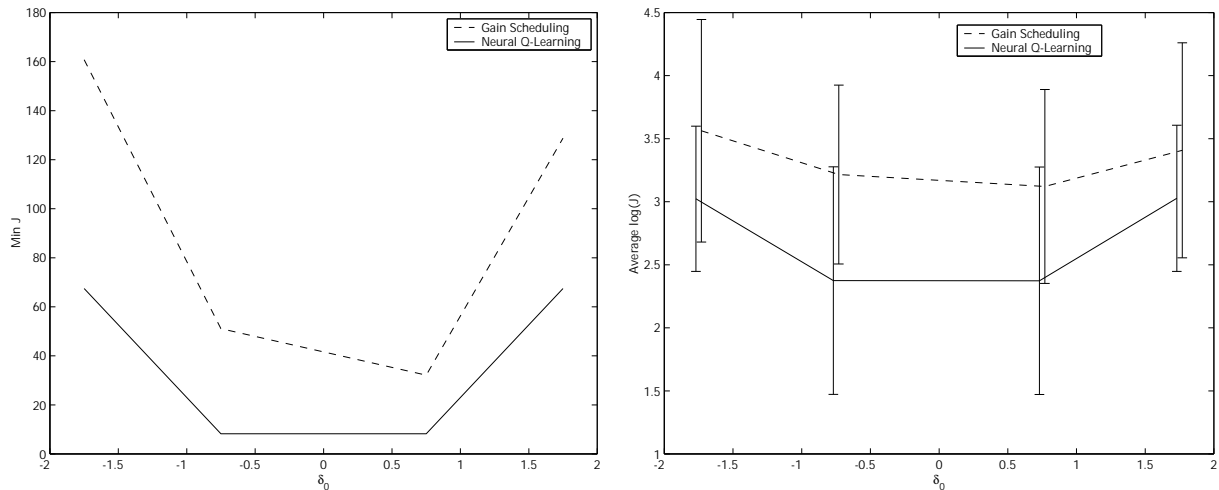
### 5.5.3 The performance of the global feedback functions

We did 30 simulation experiments to compare the performance of the two approaches. Each experiment was performed as described in the preliminary experiments. As initial  $\delta$  we used:  $-1.75$ ,  $-0.75$ ,  $0.75$  and  $1.75$ . For the Neural Q-Learning approach, the network was trained with 5 different initial weights. The network for which the lowest temporal difference error was reached was tested.

The tests were performed by running the robot for 302 time steps. This is equivalent to driving for approximately 2 minutes. The reason for using a long test period is that we wanted the robot to visit a large part of the state space.

In figure 5.8(a) we plotted the total costs as a function of the initial  $\delta$  for the best feedbacks of the two approaches. We see that the values of the resulting total costs of the Neural Q-Learning approach are symmetric around  $\delta = 0$ . This is because of the quadratic input of the network. Also we see that it is lower than the feedback for the Gain Scheduling approach. The result of the Gain Scheduling approach is not symmetric, which indicates that the robot will not approach the line exactly. Instead it will drive at a small distance parallel to the line, which also explains why the total costs are a little higher than for the Neural Q-Learning approach.

In figure 5.8(b) we see the average total costs of both approaches for all 30 experiments. Note that we plotted the the average  $\log(J)$ , because the value of  $J$  varies between 10 and  $10^5$ . We see that the Gain Scheduling approach performs very badly on average. This is because the results are either good as shown in figure 5.8(a) or very bad. These very



(a) The best performance for Gain Scheduling and Neural Q-Learning.

(b) The average performance for Gain Scheduling and Neural Q-Learning.

**Figure 5.8.** The performances of the global feedback functions.

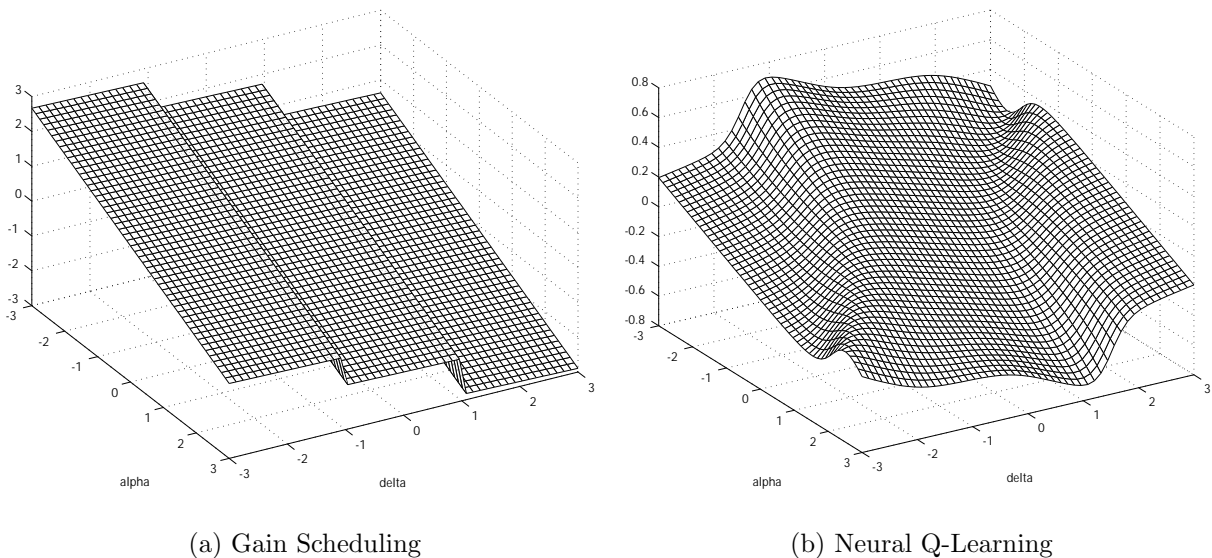
bad results are consequences of bad local feedbacks at the middle partition. If the local feedback around the line makes the robot move towards the outer partitions, the total costs will always be very high. If in this case the linear feedback in that outer partition is good, it will make the robot move towards the middle partition. As a consequence the robot gets stuck at the boundary of these two partitions.

The reason for the bad performance of Gain Scheduling is that the resulting feedback function of the extended LQRQL approach is completely determined by the train set. In chapter 4 we already showed that not all resulting feedback functions will be good. In Gain Scheduling the state space is partitioned and for each partition the train set should lead to a good function. The partitioning makes that it becomes less likely that all partitions will have a good performance. The Neural Q-Learning approach uses the complete train set. Therefore it does not have this problem, and performs much better on average.

#### 5.5.4 Training with a larger train set

The previous experiments showed that Gain Scheduling did not perform so good. The only way to improve this result is to make sure that for each partition the train set is good. We did an experiment to see whether the result of Gain Scheduling can be improved by using a larger train set for each partition. We used all the train sets from the previous experiment and combined them into one large train set. Then we applied the Gain Scheduling and Neural Q-Learning approach to this larger train set.

In figure 5.9(a) we see the resulting nonlinear feedback for Gain Scheduling. For the middle partition  $\hat{l} = -0.0032$ , which shows that it is getting closer to the correct value of



**Figure 5.9.** The control action as function of the state when a larger train set was used.

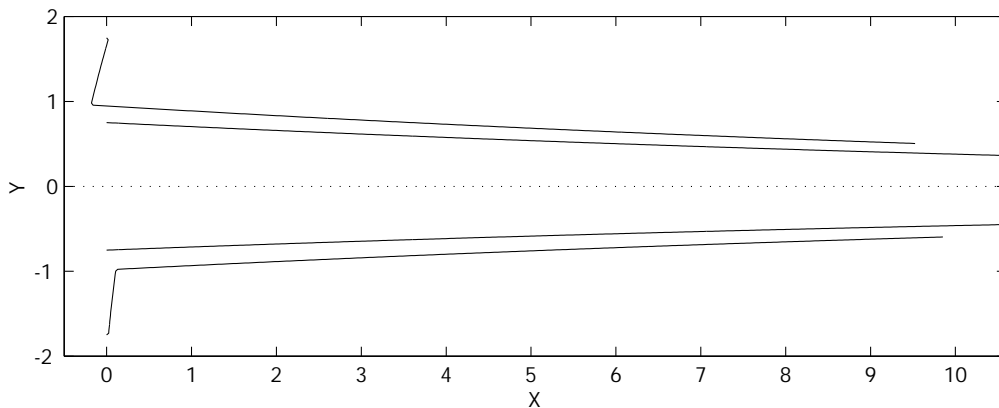
$\delta$	-1.75	-0.75	0.75	1.75
$J_{GS}$	219.3274	105.8559	91.3872	201.2774
$J_{NQ}$	121.8163	16.8742	16.8742	121.8163

**Table 5.1.** The total costs for the feedback function based on the large train set.

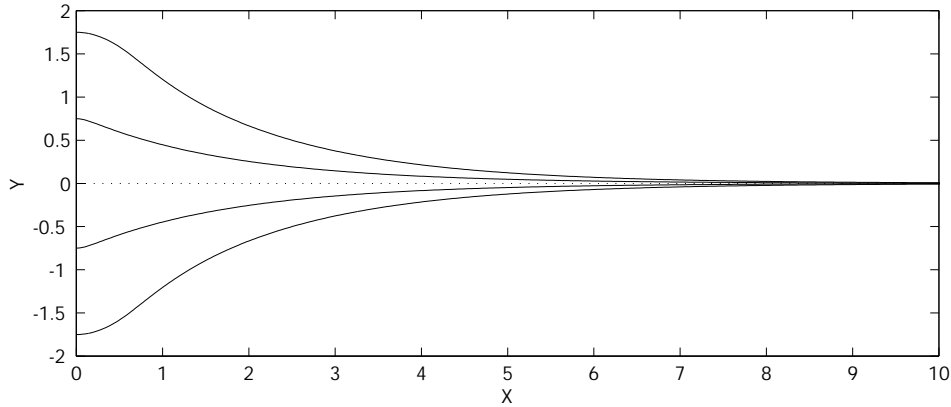
$l = 0$ . The value for  $\hat{l}$  is 0.8085 for the negative and  $-1.0222$  for the positive partition. This agrees with the discussion in section 4.5.1 where we indicated that  $L > 0$  if  $\delta < 0$  and  $L < 0$  if  $\delta > 0$ . In figure 5.9(b) we see the resulting nonlinear feedback for Neural Q-Learning. This feedback is steeper when  $\delta \approx 0$ , compared to the feedback in figure 5.6(b).

We tested the resulting feedback functions by starting the robot in the same initial states. Table 5.1 shows the resulting total costs for both approaches for different  $\delta_0$ . The total costs  $J_{GS}$  are much lower than the average value in figure 5.8(b). This shows that the increase of the train set has a huge impact on the performance of Gain Scheduling. The total costs of the neural Q-Learning approach is again lower than that of Gain Scheduling. The performance in figure 5.8(b) was already good based on smaller train sets, therefore the increase in train set does not have such a huge impact.

The trajectories of the test runs are shown in figure 5.10 and figure 5.11. In figure 5.10 we see that for the outer partitions the robot rotates  $\frac{1}{2}\pi$  in the direction of the line within one step. Then it arrives at a state where the control action is almost zero, so the robot moves straight ahead until it enters the middle partition. In the middle partition the robot will slowly approach the line. This is because the linear function of the middle partition in figure 5.9(a) is not steep enough. The reason is that for the middle partition, the feedback  $L$  hardly contributes to the control action, so that it is very difficult to estimate the quadratic Q-function. This implies that the total costs are higher than those of the neural Q-Learning approach, because of the feedback in the middle partition. The trajectories of the Neural Q-Learning approach in figure 5.11 shows that the robot moves faster to the line than the trajectories in figure 5.7(b). This is because the feedback in figure 5.9(b) is steeper near  $\delta \approx 0$ .



**Figure 5.10.** The trajectories for the Gain Scheduling feedback based on the large train set.



**Figure 5.11.** The trajectories for the Neural Q-Learning feedback based on the large train set.

## 5.6 Experiment on a Real Nonlinear System

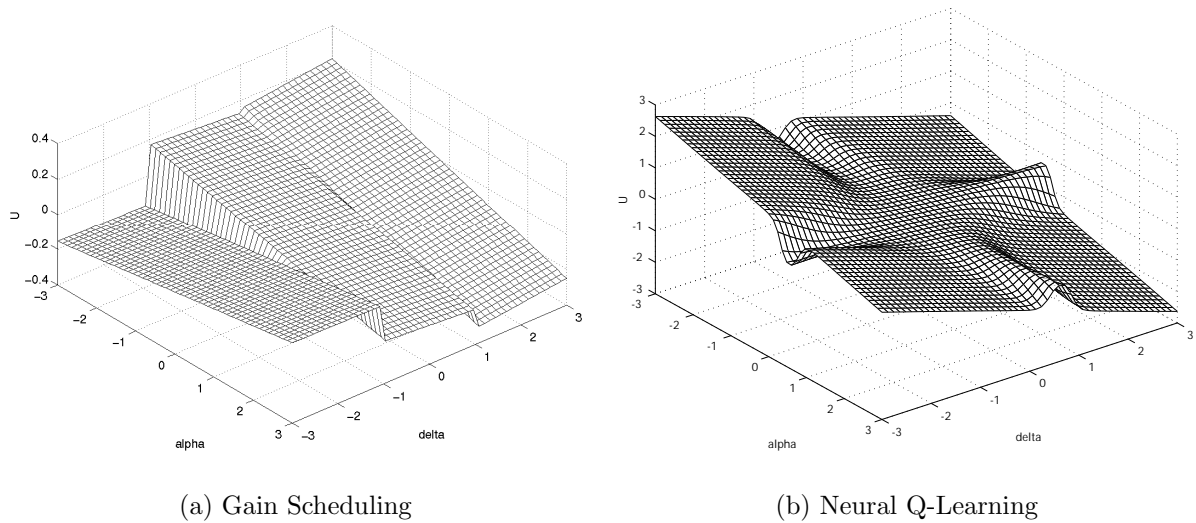
We did an experiment with the real robot, where we used the exploration noise sequence that gave the best results for both LQRQL approaches in chapter 4. The resulting feedback functions are shown in figure 5.12. In figure 5.12(a) it is clear that the local linear feedback for the partition, with the negative initial  $\delta$ , is wrong. This indicates that the exploration noise sequence used, only results in a useful training set for positive initial  $\delta$ . Since the resulting feedback in this partition is completely determined by the train set, the only way to solve this is by generating a new training set for this partition.

In figure 5.12(b) we see that the result of the Neural Q-Learning approach is again linear with a smooth nonlinear correction. Also we see that the control actions are large (we observed the same in chapter 4 for the SI approach). In fact, the linear feedback derived from the network is too large. We see that in some parts of the state space the control action is reduced by the nonlinear correction. At these states the control actions are between the action bounds of the real robot, so these actions can be applied.

Because of the space limitation of the room, we tested only for one minute. The resulting total costs are shown in Table 5.2. Again we see that the resulting total costs for the Gain Scheduling approach are not symmetric. Also we see that the costs for  $\delta_0 = -1.75$  is very high. This corresponds to the partition for which the local linear feedback is wrong. The results of the Neural Q-Learning approach show that it performs very well for all initial  $\delta$ . In order to understand the results in Table 5.2, we plotted the trajectories. In (5.13(a))

$\delta$	-1.75	-0.75	0.75	1.75
$J_{GS}$	634.007	89.008	29.993	103.707
$J_{NQ}$	82.362	11.600	11.471	82.189

**Table 5.2.** The total costs for the real nonlinear system

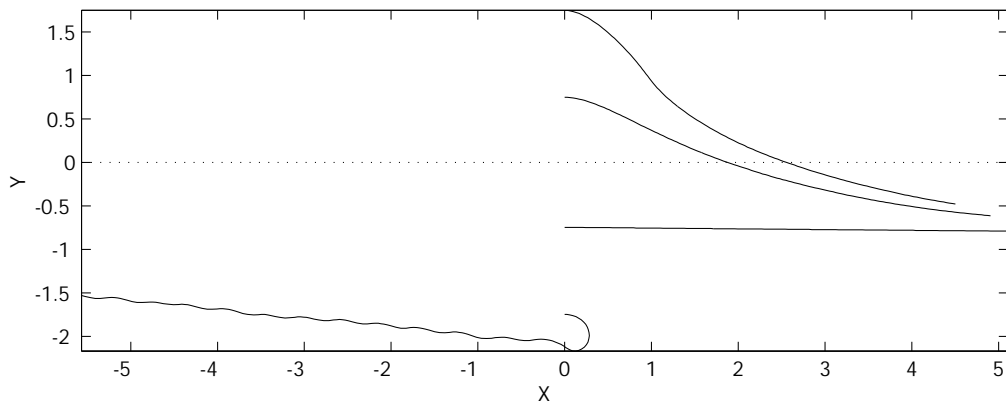


**Figure 5.12.** The control action as function of the state for the real nonlinear system.

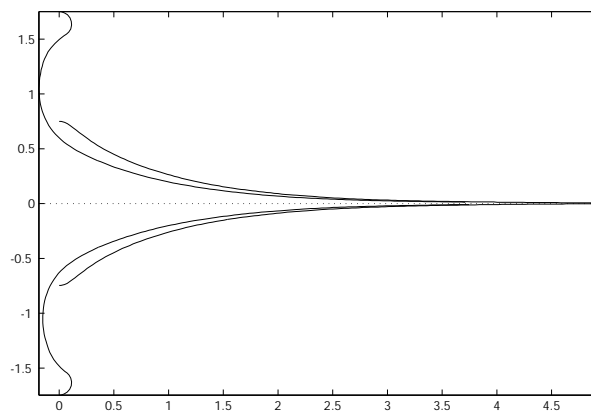
we see that for the negative partition the robot moves in the wrong direction. For this partition the local feedback is wrong. For the other partition the robot moves toward a line parallel to the line to follow. This indicates that it will never follow the line exactly. This is a consequence of the local feedback at the middle partition.

In (5.13(b)) we see that the two trajectories that start close to the line will approach the line. We also see that the trajectories far from the line first start to rotate very fast towards the line. The first few time steps the action is higher than the safety limit, so we only let the robot rotate with maximum speed. After that all actions are within the safety limit and the robot starts following the line. The high linear feedback from the network makes the robot move to the line very efficiently, when it is close to the line. The linear feedback will result in control actions that are too large far from the line. Due to the nonlinear correction the size of these actions are reduced. In figure 5.12(b) we see that this correction is for state values that were part of the train set. For other parts of the state space the control actions are still too large.

These experiments have shown that the Gain Scheduling approach depends very much on the training sets for each of the local linear feedbacks. It is possible that for some parts of the state space the feedback function found is not appropriate. The Neural Q-Learning approach results in a feedback function that is linear in most parts of the state space. In parts of the state space where the training set indicates that a correction is required, the control action deviates from the linear feedback. The resulting feedback function of Neural Q-Learning is a smooth function, unlike the feedback function of the Gain Scheduling approach where there can still be discontinuities.



(a) Trajectories using Gain Scheduling



(b) Trajectories using Neural LQRQL

**Figure 5.13.** The trajectories for the real nonlinear system.

## 5.7 Discussion

In this chapter we proposed a method to apply Q-Learning for general Q-functions. We described how a feed-forward network can be used to approximate the Q-function. The result is that the resulting feedbacks can also be nonlinear. We showed how the LQRQL approach can be used to obtain the feedback from the approximated Q-function. Then only one function has to be approximated, the feedback follows directly from the Q-function.

The advantage of using only one network is that only one network has to be trained. This makes this approach more convenient to use. More important is that the feedback function directly follows from the approximated Q-function. This is similar to Q-learning where the greedy policy is obtained by selection the greedy action for each state.

A second advantage of our approach is that the resulting feedback function is smooth. This means that there are no sudden jumps as in Gain Scheduling. These jumps can also appear when the greedy action is directly computed from the function approximator. The problem with these jumps is that the behavior of the system becomes more unpredictable in the presence of noise. The effect of the jumps can be overcome, but for Neural Q-Learning this is not necessary.

Both the Gain Scheduling approach and the Neural Q-Learning approach derive a global nonlinear feedback function from a set of linear feedback functions. In Gain Scheduling only one local linear feedback determines the control action, so it is essential that for every partition the feedback function is good. This is not always the case, resulting sometimes in very poor performances. The only way to solve this is to generate more data.

The feedback function of the Neural Q-Learning approach can be seen as a locally weighted combination of linear feedback functions. Therefore the result is less dependent on the correctness of each linear feedback function. This means that a poor performance as that of Gain Scheduling becomes very unlikely. Because the training is based on the complete train set, it will result in a good feedback function for a smaller train set than the Gain Scheduling approach.

## 5.8 Conclusions

There are different ways to get a feedback when using a feed forward network as a Q-function. Our method is based on the idea that there should be a direct relation between the Q-function and the feedback. The method uses LQRQL to obtain a linear feedback from the Q-function. Then the linear feedback is used to compute the nonlinear feedback function. The result is a global linear function with local nonlinear corrections.



# Chapter 6

## Conclusions and Future work

The objective of the research described in this thesis is to give an answer to the question whether Reinforcement Learning (RL) methods are viable methods to obtain controllers for systems with a continuous state and action space. RL is normally applied to systems with a finite number of discrete states and possible actions. Many real systems have continuous state and action spaces, so algorithms for discrete state and action spaces cannot be applied directly.

Some applications of RL to control real systems have been described [65][59][2][61][47]. This indicates that RL can be used as a controller design method. For these applications the training was performed mainly in simulation, which means that the system has to be known. However, one of the characteristics of RL is that it optimizes the controller based on interactions with the system, for which no knowledge about the system is needed. When the training is performed in simulation the potential of RL is not fully exploited.

There are two main reasons for not applying RL directly to optimize the controller of a system. The first reason is that for real control tasks a *reliable* controller is needed. The RL algorithms used for the control of continuous state space tasks are either based on heuristics or on discrete RL methods. This means that the closed loop performance when RL is applied to a real system is not very well understood. Stability during learning cannot be guaranteed, which explains the hesitation in applying these algorithms directly on real systems.

The second reason is that RL may train very slowly. In simulation the time required to obtain a sufficiently large train set depends on the computing speed. The computation of one state transition can be much faster than the actual sample time on the real system. On a real system, the time required to obtain a sufficiently large train set is determined by the system. The only way to speed this up is to use algorithms that require less training.

In this thesis we looked at Q-Learning, a model-free RL method. We investigated the applicability of Q-Learning for real systems with continuous state and action spaces. We therefore focused on the two issues mentioned above: how can we guarantee convergence in the presence of system noise and how can we learn from a small train set. We developed three methods: LQRQL for linear systems with quadratic costs, Extended LQRQL for local approximations for nonlinear systems and Neural Q-Learning for finding a nonlinear

feedback for nonlinear systems.

## 6.1 LQR and Q-Learning

A well known framework in control theory is the Linear Quadratic Regularization (LQR) task. It is an optimal control task in which the system is linear and the direct costs are given by a quadratic function of the state and action. The objective is to minimize the total future cost. When the parameters of the system are known, the optimal feedback function can be obtained by solving the Discrete Algebraic Riccati Equation. This solution is used to compute the optimal linear feedback.

When the parameters of the linear system are unknown, System Identification (SI) can be used to estimate the parameters. The estimated parameters of the system can be used to compute the optimal feedback. It has been shown [16][44] that the LQR task can be solved by Q-Learning. For this the original Q-Learning algorithm is adapted so that it can deal with the continuous state and action space of the linear system. We called this approach LQR Q-Learning (LQRQL).

It can be proven that the resulting linear feedback will eventually converge to the optimal feedback when sufficient exploration is used. These results only apply when there is no noise. In practice the system noise will always be present, so we investigated the influence of the system noise on the performance of the resulting linear feedbacks. We aimed at determining the amount of exploration required for a guaranteed improvement and compared this with the more traditional SI approach. For a fair comparison between these methods, we used a batch least squares estimation. So unlike the recursive least squares approach in [16], our results only depends on the train sets used.

To show the influence of the system noise on the performance we introduced the exploration characteristic, in which four types of outcomes can be distinguished for the SI and LQRQL approach. If the amount of exploration is:

- I Much too low: No feedback can be computed.
- II Too low: The resulting feedback will be the same as that used for the generation of the train set.
- III Almost sufficient: The resulting feedback can be anything, depending on the sequence of system noise and exploration.
- IV Sufficient: The parameters of the Q-function and system approach the correct value. The new feedback will be an improvement.

Only the type IV outcome is useful, so sufficient exploration is required. For this we derived that the SI approach requires at least more exploration than there is noise in the system. Furthermore we derived that the LQRQL approach requires more exploration than the SI approach.

### 6.1.1 Future work

**Eigenvalues:** The Q-function is given by a sum of positive definite quadratic functions (the reinforcements), so the correct Q-function is a positive definite quadratic function as well. This can be verified by looking at the eigenvalues of the matrix that represents the parameters of the Q-function. Negative eigenvalues are only possible if the parameters of the Q-function are *not* estimated correctly. This can only be the case for the type II and III outcome. So negative eigenvalues imply that insufficient exploration was used to generate the train set. In our experiments we always found negative eigenvalues for insufficient exploration, but we cannot guarantee that the estimated Q-function will never be positive definite for insufficient exploration. In order to use the eigenvalues as reliability measure, it still has to be proven that insufficient exploration will never result in a positive definite Q-function.

**Exploration:** The LQRQL approach requires more exploration than the SI approach. For real systems adding disturbance to the control action is not desirable, so to enhance practical applicability the required amount of exploration has to be reduced. One simple way of doing this is by using the SARSA variant of Q-Learning. Q-Learning uses the next action according to the feedback to compute the temporal difference. This does not include the exploration. In SARSA the action taken at the next time step is used and this also includes the exploration at that time step. So each exploration added to the control action is used twice. Experiments have shown that this approach only requires the same amount of exploration as the SI approach. However, SARSA introduces a bias in the estimation, making it perform worse for very high amounts of exploration. These results are only experimental and should be verified theoretically.

## 6.2 Extended LQRQL

LQRQL was developed for linear systems and not for nonlinear systems. Many control design approaches for nonlinear systems are based on local linear approximations of the nonlinear system. LQRQL can be used to obtain a local linear approximation of the optimal feedback function. To investigate the consequences of the nonlinearity it is possible to write the model of the nonlinear system as a linear system with a nonlinear correction.

The linear feedback of the SI and LQRQL approach are not always appropriate feedback functions to approximate the optimal feedback function. An additional offset can be included in the feedback function. This will allow for a better local approximation of the optimal feedback function in case the average nonlinear correction is large in the region of interest. To obtain the value of the offset, the parameters of a more general quadratic Q-function have to be estimated. We showed that these parameters can be estimated in the same way as the original standard LQRQL approach. We called this new approach the extended LQRQL approach. Our experiments on a simulated and on a real nonlinear system confirmed that the extended LQRQL approach results in a better local approx-

imation of the optimal feedback function. We can conclude that the extended LQRQL approach has to be applied if a nonlinear feedback function is based on multiple local linear approximations.

### 6.2.1 Future work

**On-line learning:** Throughout this thesis we used batch learning, which means that learning starts when the *complete* train set is available. We did this to make fair comparisons between the different methods possible. Usually RL is used as an on-line learning method. The recursive least squares approach for the LQR task in [16][44] is an on-line approach. Although our Q-function is quadratic, it can be seen as a function approximator that is linear in the parameters. For these function approximators the conditions are known for which convergence is guaranteed when on-line TD( $\lambda$ ) learning is used [29][31]. How the on-line TD( $\lambda$ ) approach compares with the recursive least squares approach is not known and should be investigated.

## 6.3 Neural Q-Learning

Function approximators have been used in RL. One approach uses the actor/critic configuration. One function called the critic is used to approximate the Q-function. The other function is called the actor and is used to approximate the feedback function. Both approximators have to be trained, where the approximation of the actor is trained based on the critic. This makes the training procedure rather tedious and the outcome is hard to analyze.

The LQRQL approach can be combined with a feed-forward neural network approximation of the Q-function. In this case there is no actor because in LQRQL the linear feedback function follows directly from the parameters of the Q-function. So only one function approximator has to be trained. We called this approach Neural Q-Learning.

To obtain a nonlinear feedback function for a nonlinear system using Neural Q-Learning, first a linear feedback has to be determined. The derivatives with respect to the inputs of the network give the parameters that would be estimated by the LQRQL approach. These parameters depend on the state and control action, and therefore it is not possible to directly derive a nonlinear feedback function. It is possible to ignore the tangent hyperbolic activation functions of the hidden units to obtain a globally valid linear feedback function. The linear feedback function can be used to compute the control action that is necessary to determine the nonlinear feedback function. The resulting nonlinear feedback function can be regarded as a locally weighted function, where each hidden unit results in a local linear feedback. The state value determined the weighting of these local linear function.

Experiments were performed on a simulated and on a real nonlinear system, where the goal of the experiments was to obtain global nonlinear feedback functions. We compared Neural Q-Learning with Gain Scheduling. Gain Scheduling can be performed by making for local partitions in the state space a local approximation using the extended LQRQL

approach. The experiments have shown that the neural Q-learning approach requires less training data than the Gain Scheduling approach. Therefore Neural Q-Learning is better suited for real systems.

### 6.3.1 Future Work

**Value Iteration:** The neural Q-Learning approach we presented, is still based on policy iteration. The original Q-Learning approach is based on value iteration. Because the feedback function directly follows from the approximated Q-function it is now also possible to apply value iteration. This then can be combined with an online learning approach. Whether Neural Q-Learning using Value Iteration is better than Policy Iteration still should be investigated.

**Regularization:** The weights of the hidden units should not be too large. In supervised learning there exist approaches for regularization. One way is to assign costs to the size of the weights. The consequences of this modification of the learning for Neural Q-Learning are unknown. It could have the effect that the resulting nonlinear feedback function no longer performs well. This means that further research is necessary to determine whether regularization can be applied.

## 6.4 General Conclusion

Reinforcement Learning can be used to optimize controllers for real systems with continuous state and action spaces. To exploit the benefits of reinforcement learning the approach should be based on Q-Learning, where the feedback function directly follows from the approximated Q-function. The Q-function should be represented by a function approximator, of which the parameters are estimated from the train set. If it is known that the system is linear, then the parameters of a quadratic Q-function can be estimated. This is the LQRQL approach. In case there is no knowledge about the system, the more general Neural Q-Learning approach can be applied. This will give a globally valid linear feedback function with a local nonlinear correction.



# Appendix A

## The Least Squares Estimation

### A.1 The QR-Decomposition

Two matrices  $Z$  and  $M$  have to be found for which  $ZM = X$  and  $Z^T Z = I$ . Let  $X$  have  $n$  columns and  $N - 1$  rows, then  $Z$  also has  $n$  columns and  $N - 1$  rows.  $M$  is a  $n \times n$  upper triangular matrix. Matrices  $Z$  and  $M$  can be found by Gram-Schmidt orthogonalization, for which the result can be written using projection matrices  $P$ . Let  $z_{*i}$  be the  $i^{\text{th}}$  column of  $Z$  and  $X_{*i}$  be the  $i^{\text{th}}$  column of  $X$ . Then  $z_{*i}$  is given by:

$$z_{*i} = \frac{P_i X_{*i}}{\|P_i X_{*i}\|_2}. \quad (\text{A.1})$$

Matrix  $M$  is given by:

$$M = \begin{bmatrix} \|P_1 X_{*1}\|_2 & z_{*1}^T X_{*2} & \cdots & z_{*1}^T X_{*n} \\ 0 & \|P_2 X_{*2}\|_2 & \cdots & z_{*2}^T X_{*n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \|P_n X_{*n}\|_2 \end{bmatrix}. \quad (\text{A.2})$$

The projection matrices can be defined recursively. Let  $P_1 = I$ , then for every  $j > 1$ :

$$P_j = \prod_{i=0}^{j-1} (I - z_{*i} z_{*i}^T) = I - \sum_{i=0}^{j-1} z_{*i} z_{*i}^T \quad (\text{A.3})$$

$$= I - \sum_{i=0}^{j-1} \frac{P_i X_{*i} X_{*i}^T P_i^T}{\|P_i X_{*i}\|_2^2} = P_{j-1} - \frac{P_{j-1} X_{*j-1} X_{*j-1}^T P_{j-1}^T}{\|P_{j-1} X_{*j-1}\|_2^2}. \quad (\text{A.4})$$

The projection matrix has the following properties:  $P_i^T = P_i$ ,  $P_i^2 = P_i$  and  $P_i P_j = P_j \forall j > i$ .  $P$  is a  $N - 1 \times N - 1$  matrix.

### A.2 The Least Squares Solution

The main difficulty in solving (3.30) is the computation of  $M^{-1}$ . But  $M$  is upper triangular, so the inverse can be found by backward substitution. Let  $m_{i,j}$  indicate the element of  $M$

at  $i, j$  and let  $m_{i,j}^{(-1)}$  indicate the element of  $M^{-1}$  at  $i, j$ . The elements of  $M^{-1}$  are given by:

$$m_{i,j}^{(-1)} = \begin{cases} \frac{-1}{m_{j,j}} \sum_{k=i}^{j-1} m_{k,j} m_{i,k}^{(-1)} & \text{for } i < j \\ \frac{1}{m_{i,i}} & \text{for } i = j \\ 0 & \text{for } i > j. \end{cases} \quad (\text{A.5})$$

With this result, the value of  $M^{-1}Z^T$  can be written as:

$$F = M^{-1}Z^T = \begin{bmatrix} m_{1,1}^{(-1)} z_{*1}^T + m_{1,2}^{(-1)} z_{*2}^T + \cdots + m_{1,n}^{(-1)} z_{*n}^T \\ m_{2,2}^{(-1)} z_{*2}^T + \cdots + m_{2,n}^{(-1)} z_{*n}^T \\ \vdots \\ m_{n,n}^{(-1)} z_{*n}^T \end{bmatrix}. \quad (\text{A.6})$$

Now the values of (A.5) and (A.2) should be filled in to find the expression for the rows of  $F$ . Let  $F_{*i}$  be the  $i^{\text{th}}$  row of  $F$  (with  $i < n$ ) then this can be written as:

$$F_{*i} = m_{i,i}^{(-1)} z_{*i}^T + \sum_{j=i+1}^n m_{i,j}^{(-1)} z_{*j}^T \quad (\text{A.7})$$

$$= \frac{z_{*i}^T}{m_{i,i}} + \sum_{j=i+1}^n \sum_{k=i}^{j-1} m_{k,j} m_{i,k}^{(-1)} z_{*j}^T \quad (\text{A.8})$$

$$= \frac{z_{*i}^T}{m_{i,i}} + m_{i,i+1}^{(-1)} z_{*i+1}^T + m_{i,i+2}^{(-1)} z_{*i+2}^T + \cdots + m_{i,n}^{(-1)} z_{*n}^T \quad (\text{A.9})$$

$$= \frac{z_{*i}^T}{m_{i,i}} - \frac{m_{i,i+1} z_{*i+1}^T}{m_{i,i} m_{i+1,i+1}} - \frac{z_{*i+2}^T}{m_{i+2,i+2}} \left( \frac{m_{i,i+2}}{m_{i,i}} - \frac{m_{i,i+1} m_{i+1,i+2}}{m_{i,i} m_{i+1,i+1}} \right) + \cdots \quad (\text{A.10})$$

$$= \frac{z_{*i}^T}{m_{i,i}} - \frac{z_{*i}^T X_{*i+1} z_{*i+1}^T}{m_{i,i} m_{i+1,i+1}} - \frac{1}{m_{i+2,i+2}} \left( \frac{z_{*i}^T X_{*i+2}}{m_{i,i}} - \frac{z_{*i}^T X_{*i+1} z_{*i+1}^T X_{*i+2}}{m_{i,i} m_{i+1,i+1}} \right) z_{*i+2}^T + \cdots \quad (\text{A.11})$$

$$= \frac{z_{*i}^T}{m_{i,i}} \left( I - \frac{X_{*i+1} z_{*i+1}^T}{m_{i+1,i+1}} - \frac{X_{*i+2} z_{*i+2}^T}{m_{i+2,i+2}} + \frac{X_{*i+1} z_{*i+1}^T X_{*i+2} z_{*i+2}^T}{m_{i+1,i+1} m_{i+2,i+2}} + \cdots \right) \quad (\text{A.12})$$

$$= \frac{z_{*i}^T}{m_{i,i}} \prod_{j=i+1}^n \left( I - \frac{X_{*j} z_{*j}^T}{m_{j,j}} \right) \quad (\text{A.13})$$

$$= \frac{X_{*i}^T P_i^T}{\|P_i X_{*i}\|_2^2} \prod_{j=i+1}^n \left( I - \frac{X_{*j} X_{*j}^T P_j^T}{\|P_j X_{*j}\|_2^2} \right) \quad (\text{A.14})$$

$$= \frac{X_{*i}^T P_i^T}{\|P_i X_{*i}\|_2^2} \sum_{j=i+1}^n (I - X_{*j} F_{*j}). \quad (\text{A.15})$$

For  $i = n$  there is no sum in (A.7), so the resulting expression does not have the product. The last row of the least squares solution  $\hat{\theta}_{*n}$  is given by  $F_{*n} Y$ , which results in:

$$\hat{\theta}_{*n} = \frac{X_{*n}^T P_n^T}{\|P_n X_{*n}\|_2^2} Y. \quad (\text{A.16})$$

Starting with the last row all other row can be computed recursively:

$$\hat{\theta}_{*i} = \frac{X_{*i}^T P_i^T}{\|P_i X_{*i}\|_2^2} (Y - \sum_{j=i+1}^n X_{*j} \hat{\theta}_{*j}). \quad (\text{A.17})$$

### A.3 The SI Estimation

The solution according to (3.33) starts with the last row of  $\hat{\theta}$  so first the  $\hat{\theta}_B = \hat{B}^T$  will be shown.

$$\hat{\theta}_{B,n_u*} = \frac{\mathcal{U}_{*n_u}^T P_{n_x+n_u}^T}{\|P_{n_x+n_u} \mathcal{U}_{*n_u}\|_2^2} Y_{\text{SI}} = \frac{\mathcal{E}_{*n_u}^T P_{n_x+n_u}^T}{\|P_{n_x+n_u} \mathcal{E}_{*n_u}\|_2^2} Y \quad \text{and}, \quad (\text{A.18})$$

$$\hat{\theta}_{B,i*} = \frac{\mathcal{E}_{*i}^T P_{n_x+i}^T}{\|P_{n_x+i} \mathcal{E}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=n_x+i+1}^{n_u+n_x} \mathcal{U}_{*j} \hat{\theta}_{B,j*}). \quad (\text{A.19})$$

Here we used  $P_{n_x+i} \mathcal{U}_{*i} = P_{n_x+i} (\mathcal{X} L^T + \mathcal{E}) = P_{n_x+i} \mathcal{E}_{*i}$ , because  $P_{n_x+i}$  removes the part that is linear dependent on the previous columns of  $X_{\text{SI}}$ . Then the rows of  $\hat{\theta}_A$  can be expressed as:

$$\hat{\theta}_{A,i*} = \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{\theta}_{A,*j} - \sum_{j=1}^{n_u} \mathcal{U}_{*j} \hat{\theta}_{B,j*}) \quad (\text{A.20})$$

$$= \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{\theta}_{A,*j} - \mathcal{U} \hat{\theta}_B). \quad (\text{A.21})$$

(The expression for  $\hat{\theta}_{A,n_x*}$  does not have the first sum.) The matrices of  $\hat{B}$  and  $\hat{A}$  are the transpose of  $\hat{\theta}_B$  and  $\hat{\theta}_A$ , so  $\hat{\theta}_{B,i*} = \hat{B}_{*i}$  and  $\hat{\theta}_{A,i*} = \hat{A}_{*i}$ .

The matrices  $\mathcal{X}$  and  $\mathcal{U}$  can be used to find an estimation of the feedback because  $\mathcal{U} = \mathcal{X} L^T + \mathcal{E}$ . It is also possible to write the feedback  $L$  using  $\mathcal{X}$ ,  $\mathcal{U}$  and  $\mathcal{E}$  because  $L^T = (\mathcal{X}^T \mathcal{X})^{-1} \mathcal{X}^T (\mathcal{U} - \mathcal{E})$ . So it is also possible to write:

$$L_{*i} = \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (\mathcal{U} - \mathcal{E} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} L_{j*}). \quad (\text{A.22})$$

This can be used to write (A.21) as:

$$\hat{\theta}_{A,i*} = \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{\theta}_{A,*j}) - \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} \mathcal{U} \hat{\theta}_B \quad (\text{A.23})$$

$$= \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} \hat{\theta}_{A,*j} - \mathcal{E} \hat{\theta}_B - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} L_{j*} \hat{\theta}_B) - L_{*i} \hat{\theta}_B \quad (\text{A.24})$$

$$\hat{\theta}_{A,i*} + L_{*i} \hat{\theta}_B = \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{\text{SI}} - \sum_{j=i+1}^{n_x} \mathcal{X}_{*j} (\hat{\theta}_{A,*j} + L_{j*} \hat{\theta}_B) - \mathcal{E} \hat{\theta}_B). \quad (\text{A.25})$$

Define  $\hat{\theta}_{D,i*} = \hat{\theta}_{A,i*} + L_{*i}\hat{\theta}_B$  so that:

$$\hat{\theta}_{D,i*} = \frac{\mathcal{X}_{*i}^T P_i^T}{\|P_i \mathcal{X}_{*i}\|_2^2} (Y_{SI} - \mathcal{E}\hat{\theta}_B - \sum_{j=i+1}^{nx} \mathcal{X}_{*j} \hat{\theta}_{D,*j}) \quad (\text{A.26})$$

which represents the estimation of the closed loop.

# Appendix B

## The Mobile Robot

### B.1 The robot

The robot we used in the experiments is a Nomad Super Scout II, see figure B.1. This is a mobile robot with a two wheel differential drive at its geometric center. The drive motors of both wheels are independent and the width of the robot is 41 cm. The maximum speed is 1 m/s at an acceleration of 2 m/s<sup>2</sup>.

The robot has a MC68332 processor board for the low level processes. These include sending the control commands to the drive motors, but also keeping track of the position and orientation of the robot by means of odometry. All other software runs on a second board equipped with a Pentium II 233Mhz processor.



**Figure B.1.** The Nomad Super Scout II

## B.2 The model of the robot

The mobile robot has three degrees of freedom. These define the robot's position  $x$  and  $y$  in the world and the robot's orientation  $\phi$ . The speeds of the left and right wheel  $v_l$  and  $v_r$  are the control actions that make the robot move.

Because the geometric center of the robot is right between the wheels, the control actions can also be indicated by a traversal speed  $v_t$  and rotational speed  $\omega$ . The traversal speed is given by:

$$v_t = \frac{1}{2}(v_l + v_r) \quad (\text{B.1})$$

The rotational speed is given by:

$$\omega = \frac{1}{W}(v_r - v_l), \quad (\text{B.2})$$

where  $W$  indicates the width of the robot (41 cm).

The change of of position and orientation is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \sin(\phi)v_t \\ \cos(\phi)v_t \\ \omega \end{bmatrix} \quad (\text{B.3})$$

We can notice here that the change of the orientation is independent of the position, but the change of position depends on the orientation. We also have to note that (B.3) is a simplified model of the robot, since it does not take into account the acceleration.

By taking the integral over a fixed sample time interval  $T$ , the discrete time state transition can be derived:

$$x_{k+1} = x_k + \frac{v_t}{\omega}(\sin(\phi_k + T\omega) - \sin(\phi_k)) \quad (\text{B.4})$$

$$y_{k+1} = y_k + \frac{v_t}{\omega}(\cos(\phi_k) - \cos(\phi_k + T\omega)) \quad (\text{B.5})$$

$$\phi_{k+1} = \phi_k + \omega T \quad (\text{B.6})$$

This hold for any  $\omega \neq 0$ . If  $\omega = 0$ , the orientation does not change and  $x_{k+1} = x_k + T \sin \phi_k$  and  $y_{k+1} = y_k + T \cos \phi_k$ .

# Appendix C

## Notation and Symbols

### C.1 Notation

Conventions to indicate properties of variables. The  $A$  is used as an example variable.

$A'$	The accent indicates a new result of an iteration.
$\mathbf{A}$	Bold indicates a vector.
$\hat{A}$	The hat indicates the result of an estimation.
$\tilde{A}$	The tilde indicates a dummy variable.
$A^*$	The star indicates an optimal solution.
$\bar{A}$	The bar indicates an error or deviation between the real value and the desired value.
$A_i$	A subscript can be an indication of the time step, an element form $A$ , a submatrix or the method applied. Multiple subscripts are separated by a comma.
$A_{*i}$	Indicates column $i$ of matrix $A$ .
$A_{j*}$	Indicates row $j$ of matrix $A$ .

Operations on matrices. The  $A$  is used as an example matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \quad \text{vec}(A) = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \end{bmatrix} \quad \text{vec}^\triangleleft(A) = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{22} \end{bmatrix}$$

Indices indicating the applied method.

SI	System Identification
QL	Standard LQRQL
EX	Extended LQRQL
NQ	Neural Q-Learning
GS	Gain Scheduling

## C.2 Symbols

### Chapter 1

$\mathbf{x}$	Continuous state vector
$\mathbf{u}$	Continuous control action vector
$\mathbf{v}$	Continuous noise vector, all elements are zero mean Gaussian and white
$n_x$	Dimension of the state space
$n_u$	Dimension of the control action space
$k$	Time step
$\mathbf{f}$	State transition function
$\mathbf{g}$	State feedback function
$A, B$	Parameter matrices of a linear system
$L$	Parameters of a linear state feedback
$D$	Parameters of the closed loop of a linear system

### Chapter 2

$s$	Discrete state
$a$	Discrete action
$r$	Reinforcement
$\pi$	Policy
$V$	Value function
$N$	Number of time steps
$P$	Probability matrix
$E\{\}$	Expectation value
$R$	Expected reinforcements
$\gamma$	Discount factor
$m, l$	Iteration indices
$\alpha$	Learning rate
$Q$	Q-function
$\tau$	Eligibility trace
$\xi$	Input of the function approximator
$\mathbf{w}$	Weights of the function approximator
$E$	Error function

### Chapter 3

$\sigma_v$	Standard deviation of system noise $\mathbf{v}$
$S, R$	Parameters of quadratic direct cost
$J$	Total costs
$K$	Solution of the Discrete Algebraic Riccati Equation
$e$	Exploration
$\sigma_e$	Standard deviation of exploration noise $\mathbf{e}$

$X, Y, V$	Matrices to form the least squares estimation
$\theta$	Parameters to be estimated by the linear least squares estimation
$H$	Parameters of the quadratic Q-function function, subscripts indicate submatrices of $H$
$\phi$	Concatenation of state vector $\mathbf{x}$ and control vector $\mathbf{u}$
$\rho$	Relative performance
$P$	Projection matrix
$\mathcal{X}, \mathcal{U}, \mathcal{E}$	Submatrices of $X$ for the SI approach
$c$	Some constants
$\Phi$	Matrix with difference in quadratic state/action values
$\mathbf{w}$	Noise contribution for LQRQL approach
$\mathcal{L}, \mathcal{L}_v$	Different representations of the linear feedback
$\kappa$	A constant
$\Psi$	Submatrices of $X$ for the LQRQL approach
$T, T^{ee}, \Upsilon$	Quadratic combinations of actions and exploration

#### Chapter 4

$\mathbf{w}$	Nonlinear correction
$\mathbf{x}_s$	Set point
$\mathbf{u}_s$	Action to keep system at its set point
$\mathbf{x}_{eq}$	Equilibrium state
$\mathbf{x}_{eq}$	Action at equilibrium state
$l$	Addition constant in feedback function
$G$	Extra parameters for the extended Q-function
$x, y, \phi$	Position and orientation coordinates of the robot in the world
$\delta$	Distance to the line
$\alpha$	Orientation with respect to the line

#### Chapter 5

$\mathbf{w}_o, \mathbf{w}_h, \mathbf{b}_h$	Weights and biases of the network
$\Gamma_o, \Gamma_h$	Activation functions of the units
$\Omega$	The quadratic combination of state and control action, the input for the network for Neural Q-learning



# Bibliography

- [1] P.E. An, S. Aslam-Mir, M. Brown, and C.J. Harris. A reinforcement learning approach to on-line optimal control. In *Proceedings of the International Conference on Neural Networks*, 1994.
- [2] C.G. Anderson, D. Hittle, A. Katz, and R. Kretchmar. Synthesis of reinforcement learning, neural networks, and pi control applied to a simulated heating coil. *Journal of Artificial Intelligence in Engineering*, 1996.
- [3] K.J. Åstrom and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 1989.
- [4] C.G. Atkeson, S.A. Schaal, and A.W. Moore. Locally weighted learning. *AI Review*, 1997.
- [5] C.G. Atkeson, S.A. Schaal, and A.W. Moore. Locally weighted learning for control. *AI Review*, 1997.
- [6] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.
- [7] L. Baird and A.G. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information processing Systems 11*, 1999.
- [8] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1995.
- [9] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.
- [10] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [11] D.P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, 1987.
- [12] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1997.

- 
- [13] J.A. Boyan. Least-squares temporal difference learning. In *Machine Learning: Proceedings of the Sixteenth International Conference (ICML)*, 1999.
- [14] J.A. Boyan and A.W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7 (NIPS)*, 1995.
- [15] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*. S.I.A.M., 1994.
- [16] S.J. Bradtke. Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems*, 1993.
- [17] S.J. Bradtke. *Incremental dynamic programming for on-line adaptive optimal control*. PhD thesis, University of Massachusetts, 1994.
- [18] S.J. Bradtke and A.G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 1996.
- [19] S.J. Bradtke, B.E. Ydstie, and A.G. Barto. Adaptive linear quadratic control using policy iteration. Technical Report CMPSCI 94-49, University of Massachusetts, 1994.
- [20] S.J. Bradtke, B.E. Ydstie, and A.G. Barto. Adaptive linear quadratic control using policy iteration. In *Proceedings of the American Control Conference*, 1994.
- [21] M. Campi and P.R. Kumar. Adaptive linear quadratic gaussian control: The cost-biased approach revisited. *SIAM Journal on Control and Optimization*, 1998.
- [22] H.F. Chen and L. Guo. *Identification and stochastic adaptive control*. Birkhuser, 1991.
- [23] P. Cichosz. Truncated temporal difference: On the efficient implementation of TD( $\lambda$ ) for reinforcement learning. *Journal of Artificial Intelligence Research*, 1995.
- [24] P. Dayan. The Convergence of TD( $\lambda$ ) for general  $\lambda$ . *Machine Learning*, 1992.
- [25] P. Dayan and T.J. Sejnowski. TD( $\lambda$ ) converges with probability 1. *Machine Learning*, 1994.
- [26] P.H. Eaton, D.V. Prokhorov, and D.C. Wunch II. Neurocontroller alternatives for "fuzzy" ball-and-beam systems with nonuniform nonlinear friction. *IEEE transactions on Neural Networks*, 2000.
- [27] C.N. Fiechter. PAC adaptive control of linear systems. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, 1997.
- [28] G.J. Gordon. Stable function approximation in dynamic programming. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

- 
- [29] S.H.G. ten Hagen and B.J.A. Kröse. Generalizing in TD( $\lambda$ ) learning. In *Proceedings of the third Joint Conference of Information Sciences, Durham, NC, USA*, volume 2, 1997.
- [30] S.H.G. ten Hagen and B.J.A. Kröse. A short introduction to reinforcement learning. In *Proc. of the 7th Belgian-Dutch Conf. on Machine Learning*, 1997.
- [31] S.H.G. ten Hagen and B.J.A. Kröse. Towards a reactive critic. In *Proc. of the 7th Belgian-Dutch Conf. on Machine Learning*,, 1997.
- [32] S.H.G. ten Hagen and B.J.A. Kröse. Linear quadratic regulation using reinforcement learning. In *Proc. of the 8th Belgian-Dutch Conf. on Machine Learning*,, 1998.
- [33] S.H.G. ten Hagen and B.J.A. Kröse. Pseudo-parametric Q-learning using feedforward neural networks. In *ICANN'98, Proceedings of the International Conference on Artificial Neural Networks*. Springer-Verlag, 1998.
- [34] S.H.G. ten Hagen and B.J.A. Kröse. Reinforcement learning for realistic manufacturing processes. In *CONALD 98, Conference on Automated Learning and Discovery, Carnegie Mellon University, Pittsburgh, PA*, 1998.
- [35] S.H.G. ten Hagen, D. l'Ecluse, and B.J.A. Kröse. Q-learning for mobile robot control. In *BNAIC'99, Proc. of the 11th Belgium-Netherlands Conference on Artificial Intelligence*, 1999.
- [36] K.J. Hunt, D. Sbarbaro, R. Żbikowski, and P.J. Gawthrop. Neural networks for control systems—a survey. *Automatica*, 1992.
- [37] A. Isidori. *Nonlinear control systems: An introduction*. Springer, 1989.
- [38] T. Jaakkola, M.I. Jordan, and S.P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Technical Report 9307, MIT Computational Cognitive Science, 1993.
- [39] T. Jaakkola, M.I. Jordan, and S.P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 1994.
- [40] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 1996.
- [41] H. Kimura and S. Kobayashi. An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In *proceedings of the 15th International Conference on Machine Learning*, 1998.
- [42] M.V. Kothare, V. Balakrishnan, and M. Morari. Robust constrained model predictive control using linear matrix inequalities. *Automatica*, 1996.

- 
- [43] B.J.A. Kröse and J.W.M. van Dam. Adaptive state space quantisation for reinforcement learning of collision-free navigation. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Piscataway, NJ, 1992.
- [44] T. Landelius. *Reinforcement learning and Distributed Local Model Synthesis*. PhD thesis, Linköping University, 1997.
- [45] L.J. Lin and T.M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, School of Computer Science, Carnegie Mellon University, 1992.
- [46] L. Ljung. *System Identification—Theory for the User*. Prentice Hall, 1987.
- [47] S. Miller and R.J. Williams. *Applications of Artificial Neural Networks*, chapter Temporal Difference Learning: A Chemical Process Control Application. Kluwer, 1995.
- [48] T.M. Mitchel. *Machine Learning*. McGraw Hill, 1997.
- [49] A.G. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.
- [50] R. Munos. A convergent reinforcement learning algorithm in the continuous case based on a finite difference method. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.
- [51] R. Munos. Finite-element methods with local triangulation refinement for continuous reinforcement learning problems. In *European Conference on Machine Learning*, 1997.
- [52] R. Munos. A study of reinforcement learning in the continuous case by means of viscosity solutions. *Machine Learning Journal*, 2000.
- [53] K.S. Narendra and K. Parthasarathy. Identification and control for dynamic systems using neural networks. *IEEE transaction on Neural Networks*, 1990.
- [54] H. Nijmeijer and A. van der Schaft. *Nonlinear dynamical control systems*. Springer, 1990.
- [55] J. Peng and R. J. Williams. Incremental multi-step q-learning. *Machine Learning*, 1996.
- [56] D.V. Prokhorov and D.C. Wunch II. Adaptive critic design. *IEEE transactions on Neural Networks*, 1997.
- [57] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley, 1994.
- [58] S.J. Qin and T. Badgwell. An overview of industrial model predictive control technology. In *AIChE Symposium Series 316*, 1996.

- 
- [59] M. Riedmiller. Application of sequential reinforcement learning to control dynamical systems. In *Proceedings of the IEEE International Conference on Neural Networks*, 1996.
- [60] M. Riedmiller. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. In *Neural Computing and Application Journal*, 1999.
- [61] G. Schram, B.J.A. Kröse, R. Babuska, and A.J. Krijgsman. Neurocontrol by reinforcement learning. *Journal A (Journal on Automatic Control), Special Issue on Neurocontrol*, 37, 1996.
- [62] S.P. Singh and R.S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning journal*, 1996.
- [63] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Deylon, P.Y. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 1995.
- [64] P.P. van der Smagt, F.C.A. Groen, and B.J.A. Kröse. Robot hand-eye coordination using neural networks. Technical Report CS-93-10, Dept. of Comp. Sys, University of Amsterdam, 1993.
- [65] D.A. Sofge and D.A. White. Neural network based process optimization and control. In *Proceedings of the 29th conf. on Decision and Control*, 1990.
- [66] D.A. Sofge and D.A. White. *Handbook of Intelligent Control, Neural Fuzzy, and Adaptive Approaches*, chapter Applied learning: optimal control for manufacturing. Van Nostrand Reinhold, 1992.
- [67] M. Sridhar. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning , Special Issue on Reinforcement Learning*, 1996.
- [68] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988.
- [69] R.S. Sutton. DYNA, an integrated architecture for learning, planning and reacting. In *Working Notes of the 1991 AAAI Spring Symposium on Integrated Intelligent Architectures and SIGART Bulletin 2*, 1991.
- [70] R.S. Sutton. Generalizing in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (8)*, 1996.
- [71] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

- [72] R.S. Sutton, D. McAllester, S.P. Singh, and Y Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, 2000.
- [73] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on Systems, man and Cybernetics*, 1985.
- [74] G. Tesauro. Practical issues in temporal difference learning. In *Advances in Neural Information Processing Systems 4*, 1992.
- [75] G. Tesauro. Temporal difference learning in TD-gammon. In *Communications of the ACM*, 1995.
- [76] S.B. Thrun. The role of exploration in learning control. In *Handbook of Intelligent control, Neural Fuzzy and Adaptive Approahces*. Van Nostrand Reinhold, 1992.
- [77] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, May, 1997.
- [78] E. Tzirkel-Hancock and F. Fallside. A direct control method for a class of nonlinear systems using neural networks. Technical Report CUED/F-INFENG/TR65, Cambridge University, 1991.
- [79] B. Van Roy. *Learning and Value Function Approximation in Complex Decision Processes*. PhD thesis, Massachussets Intitute of Technology, 1998.
- [80] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.
- [81] C.J.C.H. Watkins and P. Dayan. Technical note: Q learning. *Machine Learning*, 1992.
- [82] P.J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 1990.
- [83] P.J. Werbos. Approximate dynamic programming for real-time control and neural modeling. In D. A. White and D. A. Werbos Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, 1992.
- [84] M.A. Wiering and J.H Schmidhuber. Fast online  $Q(\lambda)$ . *Machine Learning*, 1998.
- [85] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.
- [86] R. Żbikowski, K.J. Hunt, A. Dzieliński, R. Murray-Smith, and P.J. Gawthrop. A review of advances in neural adaptive control systems. Technical Report ESPRIT III Project 8039: NACT, University of Glasgow/Daimler-Benz AG, 1994.

## Summary

The topic in this thesis is the use of Reinforcement Learning (RL) for the control of real systems. In RL the controller is optimized based on a scalar evaluation called the reinforcement. For systems with discrete states and actions there is a solid theoretic base and convergence to the optimal controller can be proven. Real systems often have continuous states and control actions. For these systems, the consequences of applying RL is less clear. To enhance the applicability of RL for these systems, more understanding is required.

One problem when RL is applied to real continuous control tasks is that it is no longer possible to guarantee that the closed loop remains stable throughout the learning process. This makes it a dangerous way to obtain a controller, especially since a random process called “exploration” has to be included during training. Another problem is that most RL algorithms train very slowly. This means that for a slow process the application of RL is very time consuming.

When the system is linear and the costs are given by a quadratic function of the state and control action, Linear Quadratic Regularization (LQR) can be applied. This leads to the optimal linear feedback function. System Identification (SI) can be used in case the system is unknown. The LQR task can also be solved by Q-Learning, a model-free RL approach in which the system is not explicitly modeled, but its cost function is. We called this method LQRQL. To find the optimal feedback it is necessary that sufficient exploration is used. We expressed the performance of the resulting feedback as a function of the amount of exploration and noise. Based on this we derived that a guaranteed improvement of the performance requires that more exploration is used than the amount of noise in the system. Also we derived that the LQRQL approach requires more exploration than the SI approach.

Most practical systems are nonlinear systems, for which nonlinear feedback functions are required. Existing techniques for nonlinear systems are often based on local linear approximations. The linear feedbacks of the LQRQL and SI approach are not always able to give a good local linear approximation of a nonlinear feedback function. It is possible to extend the LQRQL approach. The Extended LQRQL approach estimates more parameters and results in a linear feedback plus a constant. In an experiment on a nonlinear system we showed that the extended LQRQL approach gives a better local approximation of the optimal nonlinear feedback function.

For nonlinear systems, function approximators can be used to approximate the Q-function. We showed that it is possible to combine the LQRQL approach with a feed-forward neural network approximation of the Q-function. The nonlinear feedback function can directly be determined from the approximated Q-function, by first computing a linear feedback for which a nonlinear correction can be determined. This results in a global nonlinear feedback function. Experiments have shown that this approach requires less training data than an approach based on partitioning of the state space, where each partition approximates a linear feedback.

Reinforcement Learning can be used as a method to find a controller for real systems. To an unknown linear system the LQRQL approach can be applied. The Neural Q-Learning approach can be used when the system is nonlinear.

## Samenvatting

Het onderwerp van dit proefschrift is het gebruik van Reinforcement Learning (RL) voor het regelen van werkelijke systemen. Bij RL wordt de regelaar geoptimaliseerd op basis van een scalaire evaluatie, die reinforcement wordt genoemd. Voor systemen met discrete toestanden en regelacties is er een solide theoretische basis ontwikkeld en convergentie naar de optimale regelaar kan worden gegarandeerd. In de praktijk hebben systemen vaak continue toestanden en regelacties. Voor deze systemen zijn de consequenties van het toepassen van RL minder duidelijk. Om de toepasbaarheid van RL voor deze systemen te vergroten is meer inzicht vereist.

Een probleem dat optreedt wanneer RL wordt toegepast op werkelijke continue systemen is dat het niet meer mogelijk is om te garanderen dat het gesloten systeem stabiel blijft tijdens het leren. Dit maakt het een gevaarlijke manier om een regelaar te verkrijgen, vooral omdat ook een random proces, “exploratie” genaamd, moet worden toegevoegd tijdens het leren. Een ander probleem is dat RL algoritmes erg langzaam leren. Dit betekent dat voor een langzaam systeem het gebruik van RL erg tijdrovend is.

Wanneer een systeem lineair is en de kosten worden gegeven door een kwadratische functie van de toestand en regelactie kan Linear Quadratic Regularization (LQR) worden gebruikt. Dit levert de optimale lineaire feedback functie. Systeem Identificatie (SI) kan worden gebruikt als het systeem niet bekend is. De LQR taak kan ook opgelost worden met Q-Learning, een model-vrije RL aanpak waarin niet het systeem expliciet wordt gemodelleerd maar de kostenfunctie. We hebben deze methode LQRQL genoemd. Om de optimale feedback te vinden is het noodzakelijk om voldoende exploratie te gebruiken. We hebben de kwaliteit van de resulterende feedback beschreven als functie van de hoeveelheid exploratie en de hoeveelheid systeemruis. Op basis hiervan hebben we afgeleid dat voor een gegarandeerde verbetering van de kwaliteit het noodzakelijk is dat er meer wordt geëxploreerd dan er ruis is in het systeem. Bovendien hebben we aangetoond dat de LQRQL methode meer exploratie vereist dan de SI aanpak.

De meeste praktische systemen zijn niet-lineaire systemen, waarvoor niet-lineaire feedback functies nodig zijn. Bestaande technieken voor niet-lineaire systemen zijn vaak gebaseerd op lokaal lineaire benaderingen. De lineaire feedback van de LQRQL en de SI aanpak zijn niet altijd in staat om een goede lokale benadering te vormen van de optimale niet-lineaire feedback functie. Het is mogelijk om de LQRQL aanpak uit te bereiden. Bij de extended LQRQL aanpak worden meer parameters geschat en het resulteert in een lineaire feedback plus een constante. In een experiment met een niet-lineair systeem hebben we laten zien dat de extended LQRQL aanpak een betere lokaal lineaire benadering oplevert van de optimale niet-lineaire feedback functie.

Voor niet-lineaire systemen kunnen algemene functie schatters worden gebruikt om de Q-functie te representeren. We hebben aangetoond dat het mogelijk is om LQRQL te combineren met een feed-forward neuraal netwerk benadering van de Q-functie. De niet-lineaire feedback functie kan direct uit de benaderde Q-functie worden bepaald, door eerst een lineaire feedback te berekenen waarop een niet lineaire correctie kan worden bepaald. Het resultaat is een globale niet-lineaire functie. De experimenten hebben laten zien dat

deze aanpak minder leervoorbeelden vereist dan een aanpak gebaseerd op het partitioneren van de toestandsruimte, waarin voor iedere toestand een afzonderlijke lineaire feedback wordt bepaald.

Reinforcement Learning kan gebruikt worden als methodiek voor het vinden van regelaars voor werkelijke systemen. Voor een onbekend lineair systeem kan de LQRQL methode worden gebruikt. De Neurale Q-Learning aanpak kan worden gebruikt indien het systeem niet lineair is.

## Acknowledgments

The first people to thank are Frans Groen and Ben Kröse. Frans I want to thank for his clarity in feedback and for giving hope that this thesis would eventually be finished. Ben I want to thank for remaining a nice guy, in spite of me being sometimes a little bit stubborn, sloppy and annoying.

The project members from Delft, Bart Wams and Ton van den Boom, were very helpful with the control theoretic aspects in this thesis. Also the user group members of our STW project contributed to this thesis by indicating the importance of reliability on the practical applicability of control approaches. The work in chapter 3 was possible because Walter Hoffmann explained how the QR-decomposition can make things easier. The work on the real robot would not be possible without the help of Edwin Steffens.

I was lucky to have very entertaining room mates. Joris van Dam was always loud and funny, Anuj Dev helped me improve my ping-pong skills, Nikos Massios never was angry about my bicycle helmet jokes and Joris Portegies Zwart never stopped bragging about his favorite operating system.

Also I had colleagues with whom I enjoyed spending time. Nikos Vlassis, which is originally a Greek word, Leo Dorst, who often tried to be funnier than me, and Rien van Leeuwen, Roland Bunschoten, Sjaak Verbeek, Bas Terwijn and Tim Bouma made sure that conversations during lunch were never too seriously. I have to thank the many students as well, for providing enough distractions in the last few years. Especially Danno l'Ecluse and Kirsten ten Tusscher provided sufficient opportunities for not working on this thesis.

Last but not least, I want to thank my mother for all the lasagnas with vitamins.