REINFORCEMENT LEARNING APPLIED TO THE DOCKING OF THE AUTOMATED TRANSFER VEHICLE

by

M.J. Baars - A04011872

Delft Center for Systems and Control

A thesis submitted in partial fulfillment of the

requirements for the degree of

Master of Science

Delft 2004

TU Delft

Delft University of Technology

TABLE OF CONTENTS

LIST OF SYMBOLS

symbol	definition
$\mathbf{x}$	System state
$\mathbf{u}$	Control output
r	Immediate reward
R	Cumulative reward
V	Value function
Q	Action-value function
π	Discrete policy
α_k	Discrete learning rate
δ_k	Discrete temporal difference error
γ	Discount factor
λ	Temporal difference parameter
ϵ	ϵ-greedy parameter
δ_v	Value function temporal difference error
η_v	Value function learning rate
τ_v	Time constant for discounting future rewards
δ_m	System dynamics error
η_m	System dynamics learning rate
τ_m	Time constant for system dynamics adaptation
μ	Continuous policy
$\mathbf{e}$	Eligibility traces
κ	Time constant of the eligibility trace
σ	Exploration gain
τ_n	Time constant of the noise signal
Φ	Radial basis function
$\mathbf{w}$	Radial basis function weights
c	Radial basis function center
σ	Radial basis function width
$\mathbf{m}$	Self-organizing map reference vectors
h_{ci}	Neighborhood kernel
α	Neighborhood kernel learning rate
ϵ	Neighborhood kernel distance
σ	Neighborhood kernel width
Σ	Singular value matix
θ	Inverted pendulum angle
ω	Inverted pendulum angle velocity
μ	Inverted pendulum friction coefficient

g	Gravitational acceleration constant
e_{real}	Control error
e_{gui}	Control error from guidance
e_{nav}	Control error from navigation
e_{rtrans}	Control error from R-transform

SUMMARY

Reinforcement learning is an adaptation or learning method, that is based the same principles by which humans and animals learn to control a certain process. If an action is followed by a satisfactory state of affairs, or an improvement in the state of affairs, then the tendency to reproduce that action is strengthened, or reinforced.

Experience that has been gained over time, has to be remembered. For a process with a limited set of states and actions, this can be done in a tabular manner. For most systems, however, the state and action spaces are far too big, and alternatives have to be found to store and process the data. Several techniques exist that deal with this problem in different ways.

The currently used controllers are designed using an algorithm based on the combination of reinforcement learning and fuzzy logic, called neuro-fuzzy controllers. Gained experience is stored in the state-action table, where fuzzy logic is used to interpolate between states and actions. Besides the neuro-fuzzy method, other techniques exist, that do not rely on the principle of interpolation, but on quantization or generalization.

This project is concerned with investigating the applicability of a most promising alternative method that uses reinforcement learning in a continuous space and time framework. The gained experience is generalized by means of normalized radial basis functions. An adaptive controller has been designed, that is capable of controlling the lateral port-to-port positions and velocities during the proximity operations of the rendezvous and docking mission of the International Space Station (ISS) and the Autonomous Transfer Vehicle (ATV).

The complexity of these operations originates from the translational and rotational coupling of docking port motions and the high-frequency disturbance, caused by the flexibility of the station, that has to be compensated for. The method of reinforcement learning has been chosen as adaptation method because it does not have to make use of any physical

model for optimizing the current controller design. This could be an advantage because of the lack of reliable models for some of the disturbances.

As a means to verify the correctness of the algorithm, the controller was first applied to the inverted pendulum. Some improvements have been made to the original algorithm that improve its capabilities. The algorithm has been thoroughly tested, without and with model learning, respectively. Results of the latter are included in this thesis. The controller was then transferred to the ATV-ISS simulator, where two copies have been used in parallel.

A comparison between the neuro-fuzzy and the continuous space and time method has been made. The algorithm in its current form, does not perform as well as the neuro-fuzzy method. However, there remain several options to improve the algorithm that have not been implemented yet. Besides the simulation results, a thorough theoretical explanation of the reinforcement learning method, as well as a proposal for future research has been given.

A problem that will arise in the future, is that advanced controllers that will eventually be designed using this or related techniques, will be very intransparent to the designer. An extension to Kohonen's self-organizing map has been proposed, that addresses this problem by using principal component analysis. This technique could be used as an adaptive clustering technique, as well as its interactive monitoring, as a means to gain insight in the evolution and learning principles of the controller.

CHAPTER 1

INTRODUCTION

1.1. The rendezvous and docking mission

The rendezvous and docking/berthing mission consists of a series of orbital manoeuvres or controller trajectories, which successively bring the active vehicle (chaser) into the vicinity of, and eventually into contact with, the passive vehicle (target). The last part of the approach trajectory has to put the chaser inside the narrow boundaries of position, velocities, attitude and angular rates required for the docking process.

- In the case of *docking*, the guidance, navigation and control (GNC) system of the chaser controls the vehicle state parameters required for entry into the docking interfaces of the target vehicle and for capture.
- In the case of *berthing*, the GNC system of the chaser delivers the vehicle at nominally zero relative velocities and angular rates to a meeting point, where a manipulator, located either on the target or chaser vehicle, grapples it, transfers it to the final position and inserts it into the interfaces of the relevant target berthing port.

The complexity of the rendezvous approach and docking process and of the systems required for its execution results from the multitude of conditions and constraints which must be fulfilled.

For example, any dynamic state (position and velocities, attitude and angular rates) of the chaser vehicle outside the nominal limits of the approach trajectory could lead to collision with the target, a situation dangerous for crew and vehicle integrity. All approach trajectories must be inherently safe, even in the case of partial loss of thrust capability or control at any point of the trajectory.

The onboard system must cope with all these constraints by active control; otherwise the time-line and all events have to be pre-planned or controlled by ground. After launch, however, the nominal interaction with the spacecraft by ground is limited. For unmanned vehicles this leads to the requirement of high onboard autonomy. The combination of all the requirements, conditions and constraints, make the automatic control of rendezvous and docking/berthing by an onboard system a very complex and challenging task.

1.2. Close range rendezvous operations

The rendezvous and docking (RVD) mission basically consists of two parts: a rendezvous phase also called GPS phase and the proximity operations or rendezvous sensor (RVS) phase. During RVS phase, the docking mechanism of the chaser vehicle, has to be aligned to the docking port of the target vehicle, the ISS.

For this part of the mission, both translational and rotational motion are coupled to each other. The actual docking axis will deviate from the nominal direction due to attitude bias, attitude control motions and bending of the structure of the target vehicle. This bending results in a high frequency motion of the center of gravity of the station that causes the docking port to move up and down.

Further complications might come from internal and external disturbances working on the ATV and the ISS. Examples of these disturbances are propellant slosh of the fuel in the tanks of the ATV, the flexible motion of the ATV solar panels, the effect of the truster plumes on the target vehicle, disturbances from the truster system or environmental disturbances like there are the air drag, gravity gradient disturbance torques, or disturbance forces from the Earth gravity field. Besides these disturbances, extra complications may come from the navigation system containing measurement errors like measurement noise, biases and time delays.

For some of the disturbances, reliable models that could be used for the controller design are difficult, so not impossible, to identify. Modeling of the disturbances that could be used for robust control designs is also a very complicated task. For this reason the controller design according to the conventional methods that make use of these models could often become hard.

1.3. Objective and scope

The goal of this project is to investigate whether or not modern control approaches based on artificial intelligence and especially reinforcement learning, could be used in order to improve the performance of the controllers currently used. It is proposed to replace the original controllers, that are based on fuzzy logic and a simple form of reinforcement learning, with reinforcement learning controllers based on more novel techniques, and make a comparison with the neuro-fuzzy method, to see if there is still room for improvement.

Besides the neuro-fuzzy method, other techniques exist, that do not rely on the principle of interpolation, but on quantization or generalization. It has been decided to design the controller on the principle of generalization, because of the more recent publication on generalization and the use of reinforcement learning equations in continuous time and space [7][6], and because of a certain confidence in the author.

The evaluation of algorithms based on quantization by means of the self-organizing map [13][14] have not been included in this project, because of the limited amount of time available. It would be very interesting, however, to compare the capabilities of these three state of the art methods at a later stage.

In this project, we will use the self-organizing map, as a visualization tool for analysis and online monitoring of high dimensional functions, such as the ones used for generalization in the proposed reinforcement learning controllers. To this end, an extension to the self-organizing map is proposed, that enables us to look inside the controller during operation.

The concept of R-control [2], where the relative translational and rotational motion are decoupled, makes it possible to use an array of six independent controllers for the resulting six degrees of freedom. The array consists of four PD controllers and two neuro-fuzzy controllers, that control the lateral port-to-port positions and velocities, during the proximity operations.

Using this setup, the parallel functioning of the algorithm as well as its behavior on the ATV-ISS dynamics, can be verified using minimal computational resources. Furthermore, to make a good comparison between the neuro-fuzzy and the proposed algorithms, it was decided to keep the same setup as for the neuro-fuzzy controllers.

A detailed description of the simulator and the used models within the simulator are given in [3] [2] [4]. Because the scope of the project was emphasized on reinforcement learning rather that on the simulator design, the conditions on the simulation environment remain unchanged, that is, simplified models of the ATV-ISS dynamics have been used and besides the saw-tooth disturbance motion of the ISS and measurement noise, no other environmental disturbances are modeled. This to prevent that unnecessary energy will be spent to other factors, than the controller itself and to guarantee that a good comparison can be made. Other disturbances could be included after the goal of the project has been attained.

This thesis is structured as follows:

Chapter two gives a general introduction to reinforcement learning. Important concepts are explained in order to obtain a solid understanding of the subject as a basis for subsequent chapters. An example of how to apply the algorithms introduced is given in section (2.6). Details on the subject can be found in [15].

Chapter three explores the principle of reinforcement learning in continuous space and time as described in [7]. The equations within this chapter are slightly modified compared to the original equations, as suggested by Doya. Because they operate on the past estimates, instead of the future ones, they perform better on the control tasks at hand.

Chapter four explains the principles behind Kohonen's self-organizing map (SOM) [10] and shows how to use principal component analysis (PCA) to extend its properties so that it can be used in the analysis of high-dimensional data spaces. The modifications as such could be used to gain insight into the workings, learning principles and evolution of a multi-input multi-output (MIMO) reinforcement learning controller.

Chapter five is concerned with applying a continuous time and space reinforcement learning controller to the inverted pendulum swing-up problem. The inverted pendulum system served as a testing platform during the development phase of the project.

Chapter six elaborates on the control tasks that comprise the rendezvous and docking mission. The reader is provided with a short overview of the typical tasks, functions and system hierarchy of an automatic onboard control system for a rendezvous and docking mission, without entering into the details of the actual design.

Chapter seven elaborates on the porting of the reinforcement learning controller to the ATV system. Two identical controllers take over the role of the two fuzzy controllers, currently used on the ATV control system, and function in parallel to control the lateral relative port-to-port motion.

CHAPTER 2

INTRODUCTION TO REINFORCEMENT LEARNING

This chapter is an extract from the book of Richard S. Sutton and Andrew G. Barto on reinforcement learning. For a detailed description of the subject, the reader is advised to read the original work [15].

2.1. The reinforcement learning problem

2.1.1. Introduction

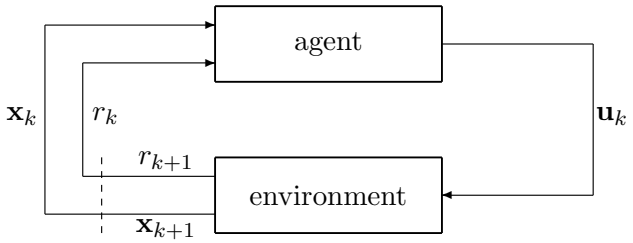

Figure 2.1: The reinforcement learning framework

Reinforcement learning is an adaptation or learning method based on human and animal learning. In this section, the full reinforcement learning problem will be introduced. The rest of this chapter deals with solving this problem. Figure 2.1 shows the *agent-environment* or the *controller-plant* interaction.

Reinforcement learning is the process of learning the *policy* through a *reward function*. The policy is a mapping of perceived states of the environment to actions to be taken when in those states. The reward function is a mapping of perceived states of the environment to a single number, the reward, indicating the intrinsic desirability of that state. It cannot be changed by the *agent*, i.e., the controller, but it can be used to change the policy.

Actions taken by the agent may affect not only the immediate reward, but also the next situation, and through that, all subsequent rewards. The total amount of reward an agent can expect to accumulate over the future starting from a particular state, is called the value of that state. The *value function* is a mapping of the perceived states of the environment to a prediction of the value of those states.

This prediction is improved upon by probing the state space for better actions, i.e. *exploration*. In contrast to exploration, the agent also utilizes experience by *exploitation* of actions that are already known to be satisfactory in obtaining future reward.

The reinforcement learning method specifies how the agent changes its policy as a result of experience, as to maximize the total reward over the long run. It uses a form of evaluative feedback rather than instructive feedback, as used in supervised learning. Evaluative feedback depends entirely on the action taken, thus creating need for active exploration, for an explicit trial-and-error search of good behavior.

The objective of reinforcement learning can be summarized as to find and optimal control strategy, the policy, that optimizes a certain control evaluation function, the value function, thereby maximizing the total amount of reward it receives, given the current policy. Reinforcement learning is based on the old idea that if an action if followed by a satisfactory state of affairs or an improvement in the state of affairs, then the tendency to reproduce that action is strengthened, i.e., reinforced.

Furthermore, time steps need not refer explicitly to fixed intervals in time, but can refer to arbitrary successive stages of decision-making and acting. Actions can be low-level controls, high-level decisions, even mental actions (e.g., shift focus of attention). States can be low-level *sensations* or more high-level and abstract, such as symbolic descriptions of objects, based on memory of past sensations, or subjective, a state of being. In general, actions can be any decision we want to learn how to make and the states can be anything we can know that might be useful for making these decisions.

The boundary between agent and environment is often not the physical boundary. In general, anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of the environment. The environment is however, not necessarily completely unknown to the agent, e.g., how rewards are computed. The agent-environment boundary represents the limit of the agents absolute control, not of its knowledge.

2.1.2. The Bellman optimality equation

All environments considered in this report are assumed to be *Markov Decision Processes* (MDP), i.e., they have the *Markov property*. This property holds if all information of the past is retained in the present state. This allows predicting the next state and expected reward through its one-step dynamics given only the current state and action.

Episodic problems have an indefinite-horizon, in which interaction can last arbitrarily long but must eventually terminate. A finite number of states and reward values enables us to work in terms of sums and probabilities rather than integrals and probability densities. *Continuous* problems have an infinite-horizon, in which interaction does not terminate. In this chapter, the episodic problem will be studied in detail. In the next chapter, we will study continuous problems. Figure 2.2 shows an episodic problem with *absorbing state* $\mathbf{x}_4$.

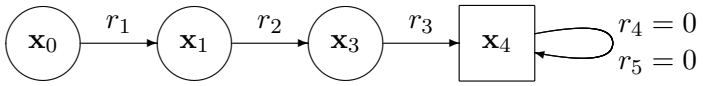

Figure 2.2: An episodic problem with absorbing state $\mathbf{x}_4$

The MDP under consideration can be written as

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \xi \tag{2.1}$$

where $\mathbf{x}_k \in \mathbf{X}$ and $\mathbf{u}_k \in \mathbf{U}$, represent the state and action at time k respectively, given the sets of possible states and actions. Equation (2.1) expresses the next state in terms of the current state and action, i.e., the one-step dynamics.

The one-step dynamics can be expressed in terms of the state transition probabilities and the reward expectations by

$$P^{\mathbf{u}}_{\mathbf{xx}'} = Pr\{\mathbf{x}_{k+1} = \mathbf{x}' | \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} \tag{2.2}$$

$$R^{\mathbf{u}}_{\mathbf{xx}'} = E[r_{k+1} | \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}, \mathbf{x}_{k+1} = \mathbf{x}'] \tag{2.3}$$

where $P^{\mathbf{u}}_{\mathbf{xx}'}$ denotes the probability of a state transition, given a possible action, and $R^{\mathbf{u}}_{\mathbf{xx}'}$ denotes the expected reward given that particular transition.

Given the sequence of rewards $r_{k+1}, r_{k+2}, \ldots$, the cumulative future reward is given by

$$R_k = r_{k+1} + r_{k+2} + \cdots + r_K \tag{2.4}$$

for an episodic problem, where K is the final time step. To assure finiteness in case of a continuous problem we write R_k as

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots \tag{2.5}$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{k+i+1} \tag{2.6}$$

where γ is the *discount factor*, $0 \leq \gamma \leq 1$. Because of the continuous nature of Chapter three, the discount factor remains to be taken into account.

The value of a state $V^{\pi}(\mathbf{x})$ is the expected return starting from that state following the current policy

$$\pi(\mathbf{x}, \mathbf{u}) = Pr\{\mathbf{u}_k = \mathbf{u} | \mathbf{x}_k = \mathbf{x}\} \tag{2.7}$$

and is given by

$$V^{\pi}(\mathbf{x}) = E_{\pi}\{R_k | \mathbf{x}_k = \mathbf{x}\} \tag{2.8}$$

$$= E_\pi\{\sum_{i=0}^{\infty}\gamma^i r_{k+i+1}|\mathbf{x}_k=\mathbf{x}\} \tag{2.9}$$

$$= E_\pi\{r_{k+1}+\gamma\sum_{i=0}^{\infty}\gamma^i r_{k+i+2}|\mathbf{x}_k=\mathbf{x}\} \tag{2.10}$$

$$= \sum_{\mathbf{u}}\pi(\mathbf{x},\mathbf{u})\sum_{\mathbf{x}'}P^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}\left[R^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}+\gamma E_\pi\left\{\sum_{i=0}^{\infty}\gamma^i r_{k+i+2}|\mathbf{x}_{k+1}=\mathbf{x}'\right\}\right] \tag{2.11}$$

$$= \sum_{\mathbf{u}}\pi(\mathbf{x},\mathbf{u})\sum_{\mathbf{x}'}P^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}\left[R^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}+\gamma V^\pi(\mathbf{x}')\right] \tag{2.12}$$

$$V^*(\mathbf{x}) = \max_\pi V^\pi(\mathbf{x}) \quad \forall\mathbf{x}\in\mathbf{X} \tag{2.13}$$

Equation (2.13) is called the *Bellman optimality equation* for the *state-value function*. For finite MDP's, this non-linear equation has a unique solution $V^*(\mathbf{x})$ for an optimal policy π^* that can be solved if we have complete and accurate knowledge of the environment dynamics and enough computational resources. The optimal policy does not have to be unique.

Similarly, we can define the value of a state-action pair as the expected reward starting from that state, taking that action and following the current policy thereafter

$$Q^\pi(\mathbf{x},\mathbf{u}) = E_\pi\{R_k|\mathbf{x}_k=\mathbf{x},\mathbf{u_k}=\mathbf{u}\} \tag{2.14}$$

$$= E_\pi\{\sum_{i=0}^{\infty}\gamma^i r_{k+i+1}|\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}\} \tag{2.15}$$

$$Q^*(\mathbf{x},\mathbf{u}) = \max_\pi Q^\pi(\mathbf{x},\mathbf{u}) \quad \forall\mathbf{x}\in\mathbf{X},\mathbf{u}\in\mathbf{U}(\mathbf{x}) \tag{2.16}$$

Equation (2.16) is called the the *Bellman optimality equation* for the *action-value function*.

The Bellman optimality equation can also be written without reference to any specific policy as

$$V^*(\mathbf{x}) = \max_{\mathbf{u}} Q^{\pi^*}(\mathbf{x},\mathbf{u}) \tag{2.17}$$

$$= \max_{\mathbf{u}} E_{\pi^*}\{R_k|\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}\} \tag{2.18}$$

$$= \max_{\mathbf{u}\in\mathbf{U}(\mathbf{x})} E_{\pi^*}\left\{\sum_{i=0}^{\infty}\gamma^i r_{k+i+1}|\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}\right\} \tag{2.19}$$

$$= \max_{\mathbf{u}\in\mathbf{U}(\mathbf{x})} E_{\pi^*}\left\{r_{k+1}+\gamma\sum_{i=0}^{\infty}\gamma^i r_{k+i+2}|\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}\right\} \tag{2.20}$$

$$= \max_{\mathbf{u}\in\mathbf{U}(\mathbf{x})} E\left\{r_{k+1}+\gamma V^*(\mathbf{x}_{k+1})|\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}\right\} \tag{2.21}$$

$$= \max_{\mathbf{u}\in\mathbf{U}(\mathbf{x})} \sum_{\mathbf{x}'}P^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}\left[R^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}+\gamma V^*(\mathbf{x}')\right] \tag{2.22}$$

Similarly, $Q^*(\mathbf{x},\mathbf{u})$ can be written as

$$Q^*(\mathbf{x},\mathbf{u}) = E\left\{r_{k+1}+\gamma\max_{\mathbf{u}'}Q^*(\mathbf{x}_{k+1},\mathbf{u}')|\mathbf{x}_k=\mathbf{x},\mathbf{u}_k=\mathbf{u}\right\} \tag{2.23}$$

$$= \sum_{\mathbf{x}'}P^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}\left[R^{\mathbf{u}}_{\mathbf{x}\mathbf{x}'}+\gamma Q^*(\mathbf{x}',\mathbf{u}')\right] \tag{2.24}$$

A graphical representation of the Bellman optimality equation can be seen in Figure 2.3. For each state or state-action pair, the action that returns the maximum reward, is chosen. This process is called a *backup operation*.

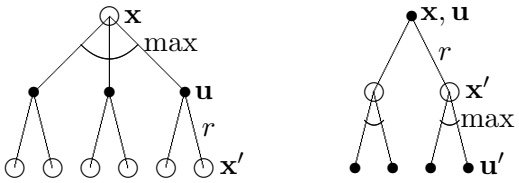

Figure 2.3: Backup diagrams for $V^*(\mathbf{x})$ and $Q^*(\mathbf{x}, \mathbf{u})$ respectively

2.2. Elementary methods for episodic tasks

2.2.1. Dynamic programming

Dynamic programming (DP) provides an essential foundation for the understanding of the methods presented in the rest of this chapter. It is however only of theoretical importance since it requires a perfect model and massive computational resources.

The value function is computed by an iterative evaluation of equation (2.12) for a given policy. Each iteration consists of a backup operation on each state. Each backup updates the value of one state based on the values of all possible successor states and their probabilities of occurring, i.e. a *full backup*. Starting at an arbitrarily chosen V_0, the sequence $\{V_k\}$ converges to V^π for $k \to \infty$.

Given the value function, an improved policy that is *greedy* with respect to V^π can be derived as

$$\pi'(\mathbf{x}, \mathbf{u}) = \arg\max_{\mathbf{u}} Q^\pi(\mathbf{x}, \mathbf{u}) > V^\pi(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbf{X} \tag{2.25}$$

The interaction of these two simultaneous processes is called the *generalized policy iteration* (GPI) and is depicted in Figure 2.4.

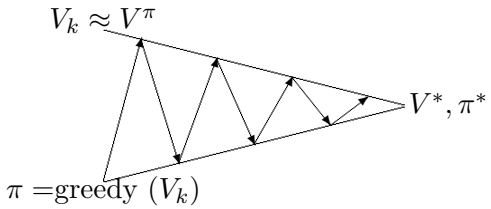

Figure 2.4: Generalized policy iteration

2.2.2. Monte Carlo

The *Monte Carlo* (MC) method does not assume complete knowledge of the environment, it requires only experience in an episode-by-episode sense. Although a model is still required, the model need only generate sample transitions, not the complete probability distributions of all possible transitions that are required by dynamic programming methods.

Only states visited for the first time in an episode will be considered, i.e. *first visit MC*. The backup operations used in MC include an entire episode, so there is no choice of actions at each state, because they have already been taken, and the estimate for one state does not build upon the estimate of any other state, as in DP. As the number of episodes increases, the value function converges to V^*.

An issue in MC is the maintaining of sufficient exploration. This can be done *on-policy* or *off-policy*. In on-policy control the agent learns about the policy that it is currently executing. The agent commits to always exploring and tries to find the best policy that still explores. In off-policy control, the *behavior policy* generates behavior in the environment, while the *estimation policy* is the policy learned about. The agent also explores, but learns a deterministic optimal policy that may be unrelated to the policy followed.

Exploration can be done by means of the the ϵ-greedy action selection method. This method exploits current knowledge to maximize the expected reward.

$$\begin{aligned} \mathbf{u}_k^* = \arg\max_{\mathbf{u}} V_{k-1}(\mathbf{x}), &\quad \text{with probability } 1-\epsilon \\ \mathbf{u}_k \neq \mathbf{u}_k^*, &\quad \text{otherwise} \end{aligned} \tag{2.26}$$

It would be even better to use an estimate of uncertainty of the action-value estimates to direct and encourage exploration. This however, is beyond the scope of this thesis.

2.2.3. Temporal difference

The *temporal difference* (TD) method is a combination of MC and DP. It learns both from experience like MC, and it *bootstraps*, i.e., estimates on the basis of other estimates, like DP.

Using equation (2.4) and (2.8), we can write down an improved estimate of the value function $V^\pi(\mathbf{x}_k)$ as

$$V(\mathbf{x}_k) \leftarrow V(\mathbf{x}_k) + \alpha_k[R_k - V(\mathbf{x}_k)] \tag{2.27}$$

$$\approx V(\mathbf{x}_k) + \alpha_k[r_{k+1} + \gamma V(\mathbf{x}_{k+1}) - V(\mathbf{x}_k)] \tag{2.28}$$

where α_k is the learning rate.

In effect, (2.27) is used for the MC update, whereas (2.28) is used for the TD update. For each time step, the estimate V^π is updated using the observed reward and the estimate $V(\mathbf{x}_{k+1})$. The term between the brackets is called the temporal difference error.

Also,

$$\sum_{k=1}^{\infty} \alpha_k = \infty \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty \tag{2.29}$$

should hold to overcome the initial condition and assure convergence respectively. Note that the $\alpha_k = \frac{1}{k}$ satisfies these conditions.

In tracking non-stationary problems, $V^*(\mathbf{x})$ changes over over time. Recent rewards should therefore be accounted for more heavily, this can be done by taking a constant step-size. As can be seen, equation (2.29) does not hold, indicating that the estimates never completely converges but continues to vary in response to the most recently received rewards. Although step-size sequences that meet these conditions are often used in theoretical work, they are seldom used in applications and empirical research.

The $V(\mathbf{x}_k)$ can be *biased* by $V_0(\mathbf{x})$ as one way to encourage exploration, i.e. *exploring starts*. Since its drive for exploration is inherently temporary, this technique is not very suitable for tracking non-stationary problems.

Now, an algorithm for evaluating the policy can be constructed in the form of the *tabular* TD(0) algorithm for estimating V^π:

Initialize $V(\mathbf{x})$ *arbitrarily,* π *to the policy to be evaluated*
Repeat (for each episode):
  Observe $\mathbf{x}$
  Repeat (for each step of episode):
    $\mathbf{u} \leftarrow$ *action given by* π *for* $\mathbf{x}$
    Take action $\mathbf{u}$*; observe reward,* r*, and next state,* $\mathbf{x}'$
    $V(\mathbf{x}) \leftarrow V(\mathbf{x}) + \alpha[r + \gamma V(\mathbf{x}') - V(\mathbf{x})]$
    $\mathbf{x} \leftarrow \mathbf{x}'$
  until $\mathbf{x}$ *is terminal*

As can be seen, the TD method does not need a model of the environment, only experience. It will find the estimates that would be exactly correct for the maximum-likely model of the Markov process given a limited set of episodes.

Policy improvement can be done in a on-policy or off-policy fashion, just as in MC. The on-policy algorithm is known as Sarsa. We will continue with the off-policy method. The off-policy TD(0) algorithm is called $Q - learning$:

Initialize $Q(\mathbf{x}, \mathbf{u})$ *arbitrarily*
Repeat (for each episode)
  Initialize $\mathbf{x}$
  Repeat (for each step of episode)
    Choose $\mathbf{u}$ *using policy derived from* Q *(e.g.,* ϵ*-greedy)*
    Take action $\mathbf{u}$*, observe* r*,* $\mathbf{x}'$
    $Q(\mathbf{x}, \mathbf{u}) \leftarrow Q(\mathbf{x}, \mathbf{u}) + \alpha[r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}') - Q(\mathbf{x}, \mathbf{u})]$
    $\mathbf{x} \leftarrow \mathbf{x}'$
  until $\mathbf{x}$ *is terminal*

As can be seen, it contains some form of GPI (section 2.2.1). The TD(0) and $Q-learning$ algorithms are *one-step tabular model free TD methods* as experience is stored in a tabular fashion, and only one-step backups are used.

2.3. Eligibility traces

In this section *eligibility traces* are introduced, as a bridge from TD to MC. They can be seen as a temporary record of the occurrence of an event.

The one-step backup for V^π is replaced by an n-step backup. In the limit case n-step TD backup equals the MC backup, the backup of an entire episode. The n-step return is introduced as

$$R_k^{(n)} = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \ldots + \gamma^{n-1} r_{k+n} + \gamma^n V_k(\mathbf{x}_{k+n}) \tag{2.30}$$

The increment to $V_k(\mathbf{x}_k)$ due to an n-step backup of $\mathbf{x}_k$ is defined as

$$\Delta V_k(\mathbf{x}_k) = \alpha_k[R_k^{(n)} - V_k(\mathbf{x}_k)] \tag{2.31}$$

Using the error reduction property of n-step returns

$$\max_{\mathbf{x}} |E_\pi\{R_k^{(n)} | \mathbf{x}_k = \mathbf{x}\} - V^\pi(\mathbf{x})| \leq \gamma^n \max_{\mathbf{x}} |V(\mathbf{x}) - V^\pi(\mathbf{x})| \tag{2.32}$$

it can be shown that n-step methods converge. Due to the inconvenience of implementation, n-step TD is rarely used.

By averaging the n-step return this we get the forward view of the TD(λ) algorithm, where

$$R_k^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_k^{(n)} \tag{2.33}$$

$$\Delta V_k(\mathbf{x}_k) = \alpha_k[R_k^\lambda - V_k(\mathbf{x}_k)] \tag{2.34}$$

As can be seen, for $\lambda = 1$, TD(λ) equals MC, for $\lambda = 0$ we get TD(0).

The forward view can be seen as follows. For each state visited, we look forward in time to all the future rewards and decide how best to combine them. After looking forward and updating one state, we move on to the next and never have to work with the preceding state again. Future states, are viewed and processed repeatedly, once from each vantage point preceding them.

Because the forward view is acausal, we need to develop a backward view. To this end, eligibility traces are introduced:

$$
\begin{aligned}
e_k(\mathbf{x}) &= \begin{cases} \gamma\lambda e_{k-1}(\mathbf{x}) & \text{if } \mathbf{x} \neq \mathbf{x}_k \\ \gamma\lambda e_{k-1}(\mathbf{x}) + 1 & \text{if } \mathbf{x} = \mathbf{x}_k \end{cases} && (2.35)\\
\delta_k &= r_{k+1} + \gamma V_k(\mathbf{x}_{k+1}) - V_k(\mathbf{x}_k) && (2.36)\\
\Delta V_k(\mathbf{x}) &= \alpha_k \delta_k e_k(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbf{X} && (2.37)
\end{aligned}
$$

On each step, the eligibility traces for all states decay by $\gamma\lambda$, and the eligibility trace for the one visited state is incremented by 1. δ_k denotes the temporal difference error.

Again, TD(λ) learning can be done on-policy or off-policy. For off-policy TD(λ) the forward view is equivalent to the backward view. However, for the off-policy TD(λ) type, called Watkins' Q(λ), the traces are cut off each time a exploratory action is taken. So the traces for all state-action pairs are either decayed by $\gamma\lambda$ or, if an exploratory action was taken, set to 0. The trace corresponding to the current state and action is incremented by 1. Using action-values, (2.35)-(2.37) become

$$
\begin{aligned}
e_k(\mathbf{x}, \mathbf{u}) &= \mathcal{I}_{\mathbf{x}\mathbf{x}_k} \cdot \mathcal{I}_{\mathbf{u}\mathbf{u}_k} + \begin{cases} \gamma\lambda e_{k-1}(\mathbf{x}, \mathbf{u}) & \text{if } Q_{k-1}(\mathbf{x}_k, \mathbf{u}_k) = \max_{\mathbf{u}} Q_{k-1}(\mathbf{x}_k, \mathbf{u}) \\ 0 & \text{otherwise} \end{cases} && (2.38)\\
\delta_k &= r_{k+1} + \gamma \max_{\mathbf{u}'} Q_k(\mathbf{x}_{k+1}, \mathbf{u}') - Q_k(\mathbf{x}_k, \mathbf{u}_k) && (2.39)\\
Q_{k+1}(\mathbf{x}, \mathbf{u}) &= Q_k(\mathbf{x}, \mathbf{u}) + \alpha_k \delta_k e_k(\mathbf{x}, \mathbf{u}) && (2.40)
\end{aligned}
$$

where $\mathcal{I}_{\mathbf{x}\mathbf{x}_k}$ and $\mathcal{I}_{\mathbf{u}\mathbf{u}_k}$ are identity-indicator functions, equal to 1 if $\mathbf{x} = \mathbf{x}_k$ or $\mathbf{u} = \mathbf{u}_k$ and equal to 0 otherwise.

The Watkins' Q(λ) algorithm is given by

Initialize $Q(\mathbf{x}, \mathbf{u})$ *arbitrarily and* $e(\mathbf{x}, \mathbf{u}) = 0$ *for all* $\mathbf{x}$,$\mathbf{u}$
Repeat (for each episode)
  Observer $\mathbf{x}$ *and initialize* $\mathbf{u}$
  Repeat (for each step of episode)
    Take action $\mathbf{u}$, *observe* r, $\mathbf{x}'$
    Choose $\mathbf{u}'$ *using policy derived from* Q *(e.g.,* ϵ*-greedy)*
    $\mathbf{u}^* \leftarrow \arg\max_{\mathbf{b}} Q(\mathbf{x}', \mathbf{b})$ *(if* $\mathbf{u}'$ *ties for the max, then* $\mathbf{u}^* \leftarrow \mathbf{u}'$*)*
    $\delta \leftarrow r + \gamma Q(\mathbf{x}', \mathbf{u}^*) - Q(\mathbf{x}, \mathbf{u})$
    $e(\mathbf{x}, \mathbf{u}) \leftarrow e(\mathbf{x}, \mathbf{u}) + 1$
    For all $\mathbf{x}$, $\mathbf{u}$
      $Q(\mathbf{x}, \mathbf{u}) \leftarrow Q(\mathbf{x}, \mathbf{u}) + \alpha\delta e(\mathbf{x}, \mathbf{u})$
      If $\mathbf{u}' = \mathbf{u}^*$, *then* $e(\mathbf{x}, \mathbf{u}) \leftarrow \gamma\lambda e(\mathbf{x}, \mathbf{u})$
        else $e(\mathbf{x}, \mathbf{u}) \leftarrow 0$
    $\mathbf{x} \leftarrow \mathbf{x}'$; $\mathbf{u} \leftarrow \mathbf{u}'$
  until $\mathbf{x}$ *is terminal*

Peng's Q(λ), which is a hybrid of Watkins' Q(λ) and Sarsa(λ), remedies the problem of cutting off traces, each time an exploratory action is taken, but is beyond the scope of this thesis.

2.4. Model based reinforcement learning

By a model of the environment we mean anything that an agent can use to predict how the environment will respond to its actions. It can be used to simulate the environment and produce simulated experience. The process that takes a model as input and produces or improves a policy for interacting with the modeled environment is called planning, or indirect reinforcement learning.

When planning is done, while interacting with the environment, new information is gained that may change the model, and thereby interact with the planning process. Within a planning agent, there are at least two roles for real experience: it can be used to improve the model and to directly improve the value function and the policy. The former we call model learning, and the latter we call direct reinforcement learning.

Figure (2.5) depicts the synergy of direct and indirect reinforcement learning, which can occur asynchronously and in parallel.

2.5. Reinforcement learning in continuous spaces

In most real world control problems, actions of a continuous nature are required in response to continuous state measurements. It should be possible that actions vary smoothly

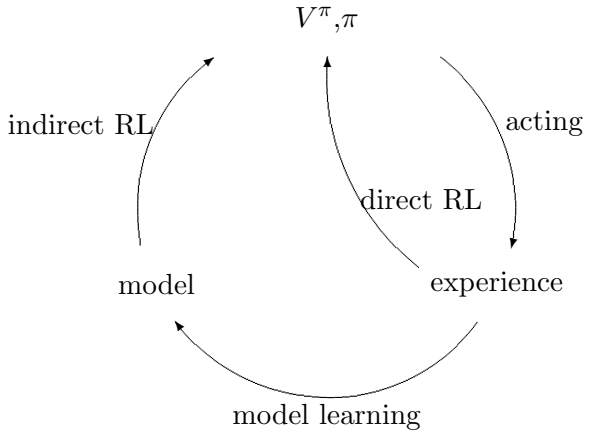

Figure 2.5: Direct vs. Indirect reinforcement learning

in response to smooth changes in state. Accurate control requires that variables be quantized finely, but as these systems fail to generalize between similar states and actions, they require large quantities of training data. Using a courser representation of states leads to aliasing, functionally different situations map to the same state and are thus indistinguishable.

The original neuro-fuzzy controllers deal with this problem, by maintaining a conventional state-action table, where fuzzy logic is used as interpolation technique.

Other techniques rely on a adaptive quantization technique by using a self-organizing map (SOM) [10]. These methods do not result in a continuous controller, but a controller with a fixed number of optimal inputs and outputs representing the continuous input and output space.

In Touzet's Q-KOHEN algorithm [16][17], a SOM implementation of QL, the neurons of the SOM represent state-actions pairs. The SOM takes the state-action pairs as inputs and produces an organized or reduced dimension expression of these input states given as vectors represented by the weights of the SOM network (Section 4.2).

Smith's *neighborhood QL* uses two separate SOM instances. The quantized state and action space represented by the SOM weights constitute a matrix-like arrangement that forms the basis needed for tabular RL techniques, e.g. TD(λ) [13][14].

The use of the SOM reduces the algorithmic learning complexity, but introduces a

extra quantization error that degrades the system performance and a computational overhead. However, for the more complex problems with more state variables, the advantage of reduction of the learning complexity becomes apparent.

A third option, is the explicit use some form of generalization, in contrast to interpolation or quantization. Several techniques that solve this problem exist [1][5][9]. They combine unsupervised reinforcement learning with function approximation methods that use supervised learning. Back propagation is used to train the function approximation network that represents the value or action-value function. In model-based learning, a second function approximation network is used to contain the system dynamics.

Frequently used function approximation methods make use of (normalized) radial basis functions (NRBF). Doya derived a set of equations for reinforcement learning in continuous time and space [7], that make use of an NRBF network. The major drawback of an NRBF network, is that it suffers from the *curse of dimensionality*, i.e. when the number of states grows linearly, computational demands grow exponentially.

This problem can be resolved by making use of the multi-layer perceptron (MLP). Although the MLP is relatively difficult to handle efficiently, Coulom did several successful experiments using Doya's equations [6]. A severe drawback of the MLP is that it suffers from slow convergence. Its computational demands, however, are much more acceptable and make it suitable for use in MIMO controllers.

2.6. Example: the random walk

The random walk [11], is a simple Markov Decision Process. The system contains five states, B through F, and two terminal states, A and G. All episodes start at state C. When an episode moves into state G, a reward of +1 occurs, all other transition rewards are 0. Transitions to the right or the left occur with equal probability.

$$A \longleftarrow B \longleftrightarrow C \longleftrightarrow D \longleftrightarrow E \longleftrightarrow F \longrightarrow G$$

This problem was analyzed using the three algorithms described in Sections 2.2 and 2.3. The following parameter settings have been used

	α	γ	ϵ	λ	episodes
TD(0)	0.1	1	-	-	1000
Q-learning	0.1	0.9	0.1	-	1000
Watkins' Q(λ)	0.15	0.9	0.1	0.9	1000

After execution, the value function for the TD(0) algorithm is given by

B	C	D	E	F
0.1854	0.4289	0.5814	0.6832	0.8348

which comes very close to

B	C	D	E	F
$\frac{1}{6}$	$\frac{2}{6}$	$\frac{3}{6}$	$\frac{4}{6}$	$\frac{5}{6}$

The state-action tables for the Q-learning and Watkins'Q(λ) respectively, are given by

	B	C	D	E	F
Q(1,x)	0.2724	0.7179	0.6632	0.7296	0.8143
Q(2,x)	0.7180	0.7302	0.8100	0.9000	1.0000

	B	C	D	E	F
Q(1,x)	0	0.5608	0.6561	0.7290	0.8099
Q(2,x)	0.6318	0.7209	0.8100	0.9000	1.0000

where action 1 represents a step to the left, and action 2 a step to the right, given the ϵ-greedy action selection.

After convergence, all episodes terminate in G, the state with the highest value. Watkins'Q(λ) converges much quicker than the conventional Q-learning, and terminates almost always correctly after only 250 episodes.

CHAPTER 3

REINFORCEMENT LEARNING IN CONTINUOUS TIME AND SPACE

3.1. Introduction

The common approach to reinforcement learning has been to discretize time, space, and action and then to apply a RL algorithm for a discrete stochastic system. A balance between fine and course discretization then has to be found. This often results in a control output that is not smooth and gives a poor performance, or in an explosion of the number of states resulting in a huge memory requirement and a large number of learning trials. To resolve this problem, a partitioning has to be found using prior knowledge.

In a continuous framework, however, a smooth control performance can be easily obtained through the use of function approximators and their gradients. There is no need to directly partition the state-space, the action-space and time. Although for a complete implementation, sampling is always required.

The algorithm for nonlinear dynamical systems presented in this chapter is based on the Hamilton-Jacobi-Bellman equation for infinite-horizon, discounted reward problems. The estimation of the value function is formulated as the minimization of a continuous-time form of the temporal difference error. The update method involves exponential eligibility traces. For policy improvement we use a value-gradient based greedy-policy [7] [6].

3.2. The reinforcement learning framework

The continuous-time system dynamics are given by

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.1}$$

where $\mathbf{x} \in \mathbf{X} \subset \mathbf{R}^n$ and $\mathbf{u} \in \mathbf{U} \subset \mathbf{R}^m$.

The immediate reward for the state and the action is:

$$r(t) = r(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.2}$$

Our goal is to find a policy

$$\mathbf{u}(t) = \mu(\mathbf{x}(t)) \tag{3.3}$$

that maximizes the cumulative future rewards

$$V^{\mu}(\mathbf{x}(t)) = \int_t^{\infty} e^{-\frac{s-t}{\tau_v}} r(s) ds \tag{3.4}$$

The value function for the optimal policy μ^* is given as

$$V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(s)\in\mathbf{U}, t\leq s<\infty} \left[\int_t^\infty e^{-\frac{s-t}{\tau_v}} r(s)ds\right] \tag{3.5}$$

According to [7] the condition for the optimal value function at time t is given by

$$\frac{1}{\tau_v}V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(t)\in\mathbf{U}} \left[r(t) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} f(\mathbf{x}(t), \mathbf{u}(t))\right] \tag{3.6}$$

which is a discounted version of the Hamilton-Jacobi-Bellman (HJB) equation. The derivation of equation (3.6) can be found in Appendix A. The optimal policy is given by the action that maximizes the right-hand side of the HJB equation, as shown in Section 2.1.2 for the Bellman optimality equation, the discrete time and space counterpart of the HJB equation

$$\mathbf{u}(t) = \mu^*(\mathbf{x}(t)) = \arg\max_{\mathbf{u}\in\mathbf{U}} \left[r(\mathbf{x}(t), \mathbf{u}(t)) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} f(\mathbf{x}(t), \mathbf{u}(t))\right] \tag{3.7}$$

In [7], the performance of several methods to estimate the value function and improving the policy is compared. For the current work, the best working methods were selected. Exponential eligibility traces are used to update the estimate of the value function (Section 3.2.1). For the policy improvement a value-gradient based greedy policy will be used (Section 3.2.2).

3.2.1. Learning the value function using exponential eligibility traces

To learn the value function in a continuous state space, a function approximation method has to be used. To this end one has to choose from several possibilities of which normalized radial basis function network (NRBF) (Appendix B) and the multi-layer perceptron (MLP) Network are the most common ones. Normalized radial basis functions will be used in this project, to enable us to exactly reproduce the experiments done by Doya [7]. A comparison of these two networks can be found Appendix C.

By differentiating equation (3.4) we obtain the consistency condition

$$\dot{V}^\mu(\mathbf{x}(t)) = \frac{1}{\tau_v}V^\mu(\mathbf{x}(t)) - r(t) \tag{3.8}$$

If the estimate of the value function equals the optimal value function, this condition will be satisfied. If not, the estimate should be adjusted to decrease the inconsistency. We define the continuous TD error

$$\delta_v(t) = r(t) - \frac{1}{\tau_v}V(t) + \dot{V}(t) \tag{3.9}$$

where $\dot{V}(t) = \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\dot{\mathbf{x}}(t)$.

Now suppose an impulse of reward is given at time $t = t_0$. Then, from equation (3.4), the resulting value function becomes

$$V^{\mu}(t) = \begin{cases} e^{-\frac{t_0 - t}{\tau_v}} & t \leq t_0 \\ 0 & t > t_0 \end{cases} \tag{3.10}$$

Because the value function is linear with respect to the reward, the desired correction of the value function for an instantaneous TD error $\delta_v(t_x)$ is

$$\hat{V}(t) = \begin{cases} \delta_v(t_0) e^{-\frac{t_0 - t}{\tau_v}} & t \leq t_0 \\ 0 & t > t_0 \end{cases} \tag{3.11}$$

We denote $\mathbf{w}_v$ the parameter of the function approximation network representing the value function. Then, the update of $w_{v,i}$ given $\delta_v(t_0)$ should be made as

$$\dot{w}_{v,i}(t) = \eta_v \int_{-\infty}^{t_0} \hat{V}(t) \frac{\partial V(\mathbf{x}(t), \mathbf{w}_v)}{\partial w_{v,i}} dt \tag{3.12}$$

$$= \eta_v \delta_v(t_0) \int_{-\infty}^{t_0} e^{-\frac{t_0 - t}{\tau_v}} \frac{\partial V(\mathbf{x}(t), \mathbf{w}_v)}{\partial w_{v,i}} dt \tag{3.13}$$

where subscript i denotes the i^{th} radial basis function (see Appendix B for details). The learning rate for the value function is denoted by η_v.

We can consider the exponentially weighted integral of the derivatives as the eligibility trace $e_i(t)$, as introduced in section (2.3), for the parameter $w_{v,i}$. Then, the algorithm to update the value function weights is derived as

$$\dot{w}_{v,i}(t) = \eta_v \delta_v(t) e_i(t) \tag{3.14}$$

$$\dot{e}_i(t) = -\frac{1}{\kappa} e_i(t) + \frac{\partial V(\mathbf{x}(t), \mathbf{w}_v)}{\partial w_{v,i}} \tag{3.15}$$

where $0 < \kappa \leq \tau_v$

3.2.2. Improving the policy

Now we consider how to improve the policy (3.3) using its associated value function V^{μ}. A greedy policy can be found by minimizing the right-hand side of our optimal policy (3.7) over a continuous set of actions at every instant. When the reinforcement signal $r(\mathbf{x}, \mathbf{u})$ is convex with respect to $\mathbf{u}$ and the system dynamics $f(\mathbf{x}, \mathbf{u})$ are linear with respect to $\mathbf{u}$ the optimization problem (3.7) has a unique solution and we can derive a closed-form expression of the greedy policy.

Given the reinforcement signal

$$r(\mathbf{x}(t), \mathbf{u}(t)) = g(\mathbf{x}(t)) - h(\mathbf{u}(t)) \tag{3.16}$$

where $g(\mathbf{x}(t))$ and $h(\mathbf{u}(t))$ represent cost functions for the state and the action respectively, then the condition for the greedy action (3.7) is given by

$$-\frac{\partial h(\mathbf{u}(t))}{\partial \mathbf{u}(t)} + \frac{\partial V(\mathbf{x}(t))}{\partial \mathbf{x}(t)} \frac{\partial f\left(\mathbf{x}(t), \mathbf{u}(t)\right)}{\partial \mathbf{u}(t)} = 0 \tag{3.17}$$

When $h(\mathbf{u}(t))$ is convex, a unique solution exists and is given by

$$\mathbf{u}(t) = -\frac{\partial h(\mathbf{u}(t))}{\partial \mathbf{u}(t)}^{-1} \left(\frac{\partial f\left(\mathbf{x}(t), \mathbf{u}(t)\right)}{\partial \mathbf{u}(t)}^{T} \frac{\partial V(\mathbf{x}(t))}{\partial \mathbf{x}(t)}^{T} \right) \tag{3.18}$$

where $-\frac{\partial h(\mathbf{u}(t))}{\partial \mathbf{u}(t)}^{-1}$ is a monotonic function. $\frac{\partial V(\mathbf{x}(t))}{\partial \mathbf{x}(t)}^{T}$ represents the steepest ascent direction of the value function (equation (B.14) - (B.17)), which is then transformed by the transpose model gain matrix $\frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}^{T}$ into a direction in the action space.

A common constraint is that the amplitude of the action, such as the force or the torque, is bounded. In equation (3.16), $h(\mathbf{u}(t))$ represents a convex cost function for the action, that can be defined as

$$h(\mathbf{u}(t)) = \mathbf{c} \int_0^{\mathbf{u}} s^{-1} \left(\frac{\mathbf{u}(t)}{\mathbf{u}^{max}} \right) d\mathbf{u} \tag{3.19}$$

where $s(\cdot)$ is a sigmoid function that saturates at $s(\pm\infty) = \pm 1$, i.e. $s(x) = \frac{2}{\pi} \arctan\left(\frac{\pi}{2} x\right)$ Then equation (3.18) becomes

$$\mathbf{u} = \mathbf{u}^{max} s \left(\frac{1}{\mathbf{c}} \frac{\partial f\left(\mathbf{x}(t), \mathbf{u}\right)}{\partial \mathbf{u}}^{T} \frac{\partial V(\mathbf{x}(t))}{\partial \mathbf{x}(t)}^{T} + \sigma(t)\mathbf{n}(t) \right) \tag{3.20}$$

where $\sigma(t)\mathbf{n}(t)$ represents the exploration noise as described in Section 3.4. $\mathbf{u}^{max}$ represents control signal bound.

3.2.3. Learning the system dynamics

There are specific cases in which the system dynamics are completely known, and therefore the model gain matrix can be derived. The model dynamics of most problems however, are not completely known, and therefore the model gain matrix has to be approximated.

When we assume $\hat{f}(\mathbf{x}(t), \mathbf{u}(t))$ the approximation of the model, the weights of the function approximation network should be updated to decrease the inconsistency

$$\delta_m(t) = f(t) - \hat{f}(t) \tag{3.21}$$

Now, if we consider the objective function

$$E(t) = \frac{1}{2} \delta_m(t)^2 \tag{3.22}$$

We denote $\mathbf{w}_m$ the parameter of the function approximator representing the system dynamics. Then, the update of $w_{m,i}$ given the objective function should be made as

$$\dot{w}_{m,i}(t) = -\eta_m \delta_m(t) \frac{\partial \delta_m(t)}{\partial w_{m,i}} \tag{3.23}$$

3.3. Discretization

In case the problem does not have continuous and differential state dynamics, the previously derived equations will have to be discretized. To this end, we use the backward Euler approximation, because it is easily implemented by hand. To update the value function and system dynamics weights we integrate again using the backward Euler approximation. The plant dynamics are integrated using the Runge-Kutta approximation.

The backward Euler approximation is defined as:

$$\dot{f}(t) = \frac{f(t) - f(t - dt)}{dt} \tag{3.24}$$

Since the boundary condition for the value function is given at $t \to \infty$ as can be seen in equation (3.4) the best course of action is to update the past estimates of the value function weights and the model weights. To accomplish this, some equations should be slightly modified compared to [7].

3.3.1. Learning the value function using exponential eligibility traces

By applying (3.26), equation (3.9) is rewritten to

$$\begin{aligned}
\delta_v(t) &= r(t) - \frac{1}{\tau_v} V(t) + \dot{V}(t) && (3.25)\\
&= r(t) - \frac{1}{\tau_v} V(t) + \frac{V(t) - V(t-1)}{dt} && (3.26)\\
&= r(t) + \frac{1}{dt}\left(\left(1 - \frac{dt}{\tau_v}\right) V(\mathbf{x}(t), \mathbf{w}_v(t - dt)) - V(\mathbf{x}(t - dt), \mathbf{w}_v(t - dt))\right) && (3.27)
\end{aligned}$$

Equation (3.12) should be integrated to $t_0 - dt$ so that (3.14) and (3.15) are given as

$$\begin{aligned}
\dot{w}_{v,i}(t) &= \eta_v \delta_v(t) e_i(t) && (3.28)\\
\dot{e}_i(t) &= -\frac{1}{\kappa} e_i(t) + \frac{\partial V(\mathbf{x}(t - dt), \mathbf{w}_v(t - dt))}{\partial w_{v,i}} && (3.29)
\end{aligned}$$

where $0 < \kappa \leq \tau_v$

Then, we can calculate the updated weights using (3.25) as

$$\begin{aligned}
e_k(t) &= e_k(t - dt) + dt \cdot \dot{e}_i(t) && (3.30)\\
w_{v,i}(t) &= w_{v,i}(t - dt) + dt \cdot \dot{w}_{v,i}(t) && (3.31)
\end{aligned}$$

3.3.2. Learning the system dynamics

Using the backward Euler approximation and substitution $f(t)$ by the state $\dot{\mathbf{x}}(t)$ we can write equation (3.19) as

$$\begin{aligned} \delta_m(t) &= f(t) - \hat{f}(t) && (3.32)\\ &\approx f(t) - \hat{f}(t - dt) && (3.33)\\ &= \dot{\mathbf{x}}(t) - \hat{f}(t - dt) && (3.34) \end{aligned}$$

We write equation (3.23) as

$$\dot{w}_{m,i}(t) = \eta_m\left(\frac{\mathbf{x}(t) - \mathbf{x}(t - dt)}{dt} - \hat{f}(t - dt)\right)\frac{\partial \hat{f}(t - dt)}{\partial w_{m,i}} \tag{3.35}$$

Then, the updated weights can be calculated as

$$w_{m,i}(t) = w_{m,i}(t - dt) + dt \cdot \dot{w}_{m,i}(t) \tag{3.36}$$

3.4. The exploration mechanism

In this section, we describe the exploration mechanism. It is given by the simple differential equation:

$$\tau_n \dot{\mathbf{n}}(t) = -\mathbf{n}(t) + \mathbf{N}(t) \tag{3.37}$$

$$\sigma(t) = \sigma_0 \min\left[1, \max\left[0, \frac{V_1 - V(t)}{V_1 - V_0}\right]\right] \tag{3.38}$$

that represents a dynamic filter, where $\mathbf{N}(t)$ denotes normal Gaussian noise, V_0 and V_1 are the minimal and maximal levels of the expected reward that are initialized as constants. τ_n is a time constant.

As for equation (3.42), it can be seen that when the controller resides in less preferable states, the exploration factor is at a maximum. As performance increases, exploration decreases.

3.5. The algorithm

The algorithm derived can be summarized as follows.

Algorithm 3.1 - Reinforcement Learning in continuous time and space

Initialize the value function and model weights, $\mathbf{w}_v$ *and* $\mathbf{w}_m$

Repeat for each trial

Initialize the eligibility traces, $\mathbf{e}_i$

Update $\mathbf{w}_v$ *and* $\mathbf{w}_m$

$$\begin{aligned} \delta_v(t) &= r(t) + \frac{1}{dt}\left(\left(1 - \frac{1}{dt}\right) V(\mathbf{x}(t), \mathbf{w}_v) - V(\mathbf{x}(t-dt), \mathbf{w}_v)\right) \\ \delta_m(t) &= \frac{\mathbf{x}(t) - \mathbf{x}(t-dt)}{dt} - f(\mathbf{x}(t-dt), \mathbf{u}(t-dt), \mathbf{w}_m) \end{aligned}$$

$$\begin{aligned} \dot{e}_i(t) &= -\frac{1}{\kappa} e_i(t) + \frac{\partial V(\mathbf{x}(t-dt), \mathbf{w}_v)}{\partial w_{v,i}} \\ \dot{w}_{v,i}(t) &= \eta_v \delta_v(t) e_i(t) \\ \dot{w}_{m,i}(t) &= \eta_m \delta_m(t) \frac{\partial f(\mathbf{x}(t-dt), \mathbf{u}(t-dt), \mathbf{w}_m)}{\partial w_{m,i}} \end{aligned}$$

$$\begin{aligned} e_i(t) &= e_i(t-dt) + dt * \dot{e}_i(t) \\ w_{v,i}(t) &= w_{v,i}(t-dt) + dt * \dot{w}_{v,i}(t) \\ w_{m,i}(t) &= w_{m,i}(t-dt) + dt * \dot{w}_{m,i}(t) \end{aligned}$$

Obtain the controller output $\mathbf{u}(t)$

$$\begin{aligned} \mathbf{u} &= \mathbf{u}^{max} s \left(\frac{1}{c} \left. \frac{\partial f(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}_m)}{\partial \mathbf{u}}^T \right|_{\mathbf{u}=0} \frac{\partial V(\mathbf{x}(t), \mathbf{w}_v)}{\partial \mathbf{x}(t)}^T + \sigma \mathbf{n}(t) \right) \\ s &= \frac{2}{\pi} \arctan\left(\frac{\pi}{2}\phi\right) \end{aligned}$$

end

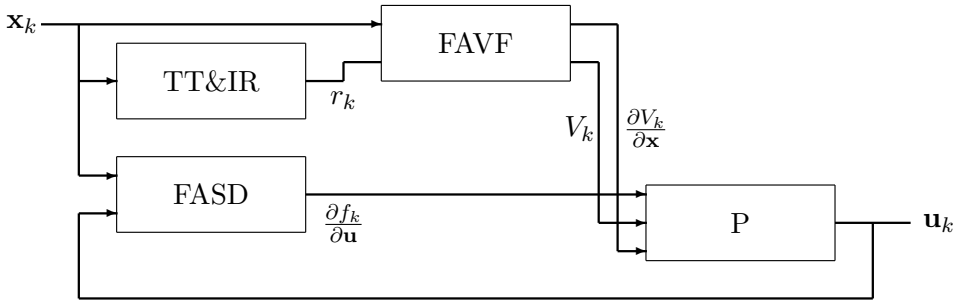

Figure 3.1: Implementation of algorithm 3.1

The implementation is depicted in Figure 3.1. The following abbreviations have been used

TT&IR	Trial Termination and Immediate Reward
FAVF	Function Approximator for the Value Function
FASD	Function Approximator for the System Dynamics
P	Policy

CHAPTER 4

THE SELF-ORGANISING MAP

4.1. The principle of the SOM

The self-organizing map (SOM) represents the result of a vector quantization algorithm that places a number of reference vectors into a high-dimensional input data space to approximate the data. During the learning process, the reference vectors are made dependent on each other as if they would lie along an *elastic surface*. By means of the self-organizing algorithm, this surface becomes defined as a kind of nonlinear regression of the reference vectors through the data points. A mapping from a high-dimensional data space onto, say, a two-dimensional lattice of points is thereby also defined. Such a mapping can effectively be used to visualize metric ordering relations of input samples. In practice, the mapping is obtained as an asymptotic state in the learning process.

A typical application of this kind of SOM is in the analysis of complex experimental vectorial data such as process states, where the data elements may even be related to each other in a highly nonlinear fashion. The process in which the SOM is formed is an unsupervised learning process. Like any unsupervised learning classification method, it may also be used to find clusters in the input data, and to identify an unknown data vector with one of the clusters. On the other hand, if the data are a priori known to fall in a finite number of classes, identification of an unknown data vector would optimally be done by some supervised learning algorithm, say, the learning vector quantization (LVQ), which is related to the SOM.

4.2. The mathematics of the SOM

There exist many versions of the SOM. The basic philosophy, however, is very simple and already effective as such. The SOM here defines a mapping from the input space $\mathbf{R}^n$ onto a regular two-dimensional array of nodes. With every node i, a reference vector $\mathbf{m}_i \in \mathbf{R}^n$ is associated. The lattice type of the array can be defined as rectangular or hexagonal, the latter is more effective for visual display. An input vector $\mathbf{x}$ is compared with the $\mathbf{m}_i$, and the best match is defined as a response: the input is thus mapped onto this location. One might say that the SOM is a nonlinear projection of the probability density function of the high-dimensional input data onto the two-dimensional display.

Let

$$\mathbf{x} \in \mathbf{R}^n, \quad \mathbf{m}_i \in \mathbf{R}^n \tag{4.1}$$

be the input data vector and the i_{th} reference vector, respectively. The smallest Euclidean distance defines the *winner node* $\mathbf{m}_c$

$$\|\mathbf{x} - \mathbf{m}_c\| = \min_i \|\mathbf{x} - \mathbf{m}_i\| \tag{4.2}$$

Useful values of $\mathbf{m}_i$ can be found as the convergence limits of the following regression, whereby the initial values $\mathbf{m}_i(0)$ can be arbitrary

$$\mathbf{m}_i(k+1) = \mathbf{m}_i(k) + h_{ci}(k)[\mathbf{x}(k) - \mathbf{m}_i(k)] \tag{4.3}$$

where k is the discrete-time coordinate and $h_{ci}(k)$ is the so-called *neighborhood kernel*. It is a function defined over the lattice points and is usually defined as

$$h_{ci}(k) = h\left(\|\mathbf{r}_c - \mathbf{r}_i\|, k\right) \tag{4.4}$$

where $\mathbf{r}_c \in \mathbf{R}^2$ and $\mathbf{r}_i \in \mathbf{R}^2$ are the position vectors of nodes c and i, representing their position on the lattice respectively. During learning, those nodes that are topographically close in the array up to a certain distance will activate each other to learn from the same input. With increasing $\|\mathbf{r}_c - \mathbf{r}_i\|$, $h_{ci} \to 0$. The average width and form of h_{ci} defines the *stiffness* of the elastic surface.

Possible definitions of h_{ci} include the *bubble* kernel and the *gaussian* kernel. The bubble kernel refers to a set of array points around node c and is defined as

$$h_{ci}(k) = \begin{cases} \alpha(k) & \text{if } \|\mathbf{r}_c - \mathbf{r}_i\| < \epsilon(k) \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

where learning rate $\alpha(k)$, $0 < \alpha(k) \leq 1$ is a monotonically decreasing function of time and $\epsilon(k)$ is the *kernel distance*.

The gaussian kernel is defined as

$$h_{ci}(k) = \alpha(k) \exp\left(-\frac{\|\mathbf{r}_c - \mathbf{r}_i\|^2}{2\sigma^2(k)}\right) \tag{4.6}$$

where $\alpha(k)$, $0 < \alpha(k) \leq 1$ is the learning rate and $\sigma(k)$ the width of the kernel. Both functions are monotonically decreasing functions of time.

The SOM introduces a quantization error. The minization of the *average quatization error*

$$\frac{1}{m}\sum_{i=1}^{m}\|\mathbf{x}_i - \mathbf{m}_c\| \tag{4.7}$$

is often used as a performance index, where N is the number of input data vectors. This performance index can be minimized by trying several different $\mathbf{m}_i(0)$.

As $h_{ci}(k)$ is shifted toward $h_{ci} = \delta_{ci}$, where δ_{ci} is the Kronecker delta, the quantization error would become minimal, but there wouldn't be self-organizing properties either. The mutual interaction between the lattice nodes through the neighborhood kernel is what makes the nodes group and organize themselves.

Furthermore, the SOM can be calibrated by manually selected data sets. It can be trained by reiteration on a limited data set and the importance of a particular data vector can be enhanced by increasing the neighborhood kernel locally. This way, one could emphasize particular area's of interest in the resulting map, or incorporate prior knowledge of the input data vectors.

4.3. Topology preserving projections

When dealing with RL controllers in continuous space and time, some form of function approximation networks are used to represent the knowledge acquired during learning. In extending these controllers to the MIMO case, these function approximation networks become high-dimensional functions, that are very difficult to interpret properly.

In order to visualize such a high-dimensional data space while retaining as much information as possible, a suitable projection onto a two or three dimensional space has to be found. To this end, an algorithm was developed that decomposes the SOM weights, the generalization of all data points, using a singular value decomposition. The data points are projected onto a suitable hyperplane that is the span of the principal components of the decomposition, then the projection is mapped onto the visualization plane. This algorithm can be easily extended to a projection in three dimensional space.

Using a singular value decomposition, the SOM weights are written as

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4.8}$$

where $\mathbf{M}, \mathbf{\Sigma} \in \mathbf{R}^{m\times n}$, $\mathbf{U} \in \mathbf{R}^{m\times m}$, and $\mathbf{V} \in \mathbf{R}^{n\times n}$, being m the number of SOM neurons, and n the dimension of the original data space, where the diagonal elements of $\mathbf{\Sigma}$

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \tag{4.9}$$

represent the first r singular values of the rank r matrix $\mathbf{M}$.

The principal component vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\subset \mathbf{V}$ are non unique orthonormal vectors that represent the unit directions in which the data variance is greatest. By selecting these vectors as a basis that spans a hyperplane in $\mathbf{R}^n$ onto which the projection of the data points is done, maximum topological information entropy is contained within this projection. The projection itself is done using the orthogonal projection theorem

$$\mathbf{p}_{hp} = \frac{\mathbf{p} \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 + \frac{\mathbf{p} \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2}\mathbf{v}_2 \tag{4.10}$$

where subscript *hp* denotes the hyperplane. The SOM weights can be projected onto the hyperplane using this theorem too.

The projected data points are mapped back onto the visualization plane using

$$\mathbf{p}_{vp} = \mathbf{p}_{hp}\mathbf{V}^{-1} \tag{4.11}$$

where subscript *vp* denotes the visualization plane. All elements of $\mathbf{p}_{vp}$ are zero, except for the first two. They can therefore be disregarded. This leaves us with a set of data points in $\mathbf{R}^2$.

The complete algorithm is given by

$\mathbf{U\Sigma V}^T = \mathbf{M}$ *; singular value decomposition*
$\mathbf{ob} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}$
for $i = 1$ *to* m
  $\mathbf{m}_{hp,i} = proj(\mathbf{m}_i, \mathbf{ob})$ *; topology preserving orthogonal projection*
  $\mathbf{m}_{vp,i} = \mathbf{m}_{hp,i}\mathbf{V}^{-1}$ *; map hyperplane onto visualization plane*
end
for $i = 1$ *to* n
  $\mathbf{p}_{hp,i} = proj(\mathbf{p}_i, \mathbf{ob})$
  $\mathbf{p}_{vp,i} = \mathbf{p}_{hp,i}\mathbf{V}^{-1}$
end

Furthermore, the total percentage of variance retained in the projection is given by the following formula

$$pv = \frac{\sigma_1^2}{\sum^r \sigma_j^2} + \frac{\sigma_2^2}{\sum^r \sigma_j^2} \tag{4.12}$$

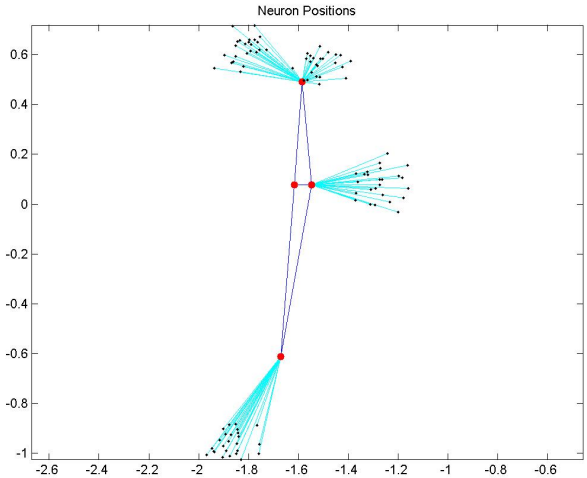

Figure 4.1: Visualization of a data and weight space in $\mathbf{R}^{10}$

Figure 4.1 shows the projection of four clusters in a ten-dimensional space, using four neurons in a hexagonal type lattice. During training, the reference vectors move toward the data vectors. As can be seen, the SOM is useful as an indication of whether possible clusters exist and where they are located. Visual verification of the clusters is made possible by the projection.

This would thus be a very interesting tool for dynamically clustering a given high-dimensional state space. The centers of a normalized radial basis function network could be placed very efficiently as a part of the reinforcement learning algorithm, while retaining the possibility of extracting valuable visual information and online monitoring of the algorithm during operation.

4.4. The projection of the value function

To obtain maximum visual information from the high-dimensional value function, the value function can be projected onto the hyperplane. The projection is done using the same orthonormal vectors that we used in projecting the weights, the boundaries being determined by the mapped visualization plane.

This is done by the following algorithm

for each $\mathbf{x}_{vp} \in \mathbf{X}_{vp}$; $\mathbf{X}_{vp} \subset \mathbf{R}^2$
  $\mathbf{x}_{hp} = x_{vp,1}\mathbf{v}_1 + x_{vp,2}\mathbf{v}_2$; $\mathbf{v}_i \in \mathbf{V}^{n \times n}$
  $V_{hp}(\mathbf{x}_{vp}) = V(\mathbf{x}_{hp})$
end

where subscript *vp* denotes the visualization plane, subscript *hp* denotes the hyperplane, and n denotes the dimension of the original data space.

In case of the projection a value function, the projection will be independent of $\mathbf{u}$. This method is easily extended to the projection of a action-value function. Vector $\mathbf{p}$ will then of the form

$$\mathbf{p}_i = \begin{bmatrix} \mathbf{x}_i \\ \mathbf{u}_i \\ Q(\mathbf{x}_i, \mathbf{u}_i) \end{bmatrix} \tag{4.13}$$

where the inclusion of $Q(\mathbf{x}_i, \mathbf{u}_i)$ is optional. When the SOM is trained with the inclusion of the value function or the action-value function, it is possible to show the projected value or action-value function in a quantized manner, by assigning a color index to the winner nodes.

Figures 4.2 and 4.3 are projections of a three-dimensional *state-value* space, using two neurons. With the inclusion of $V(\mathbf{x}_i)$, it is also assured that all elements of the original data space, as well as the value function itself, are projected with maximum preservation of visual information.

Figure 4.4 is a projection of the two-dimensional state space only. The SOM was trained using eight neurons. The value function is projected on the background, and matches the projected state space very closely.

Figure 4.5 is an attempt to visualize the state-action space, using just the value function. One would probably obtain better results if this action-value function were de-

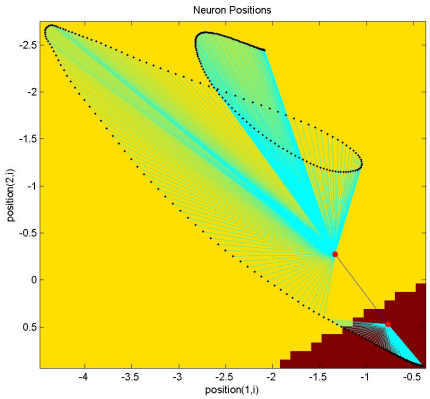

Figure 4.2: Quantized visualization of a state-value and weight space in $\mathbf{R}^3$

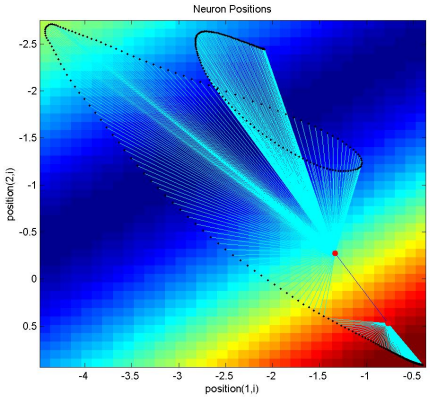

Figure 4.3: Visualization of a state-value and weight space in $\mathbf{R}^3$

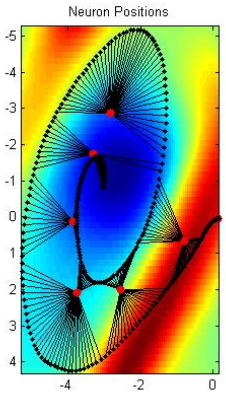

Figure 4.4: Visualization of the state and weight space in $\mathbf{R}^2$

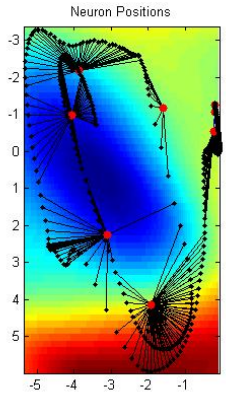

Figure 4.5: Visualization of the state-action space in $\mathbf{R}^3$

pendent on $\mathbf{u}$. The addition of this extra dimension now only introduces uncertainty in the visualization process, still, 96.6% of the visual information is contained within this figure.

This projection mechanism has a significant drawback. Although projected with maximum information entropy, the accuracy of the projected data is limited and does not coincide with projection of the value function onto the hyperplane. This means that an information shift can occur in the visualization plane, as can be seen in the lower right corner of Figure 4.2. The switching of neurons does not occur at the same visualized position. A possible solution to this problem involves interpolating the value function between the transformed data points using splines.

Furthermore, the following inequality holds for all projected data

$$\|\mathbf{p}_i - \mathbf{p}_{hp,i}\| < \|\mathbf{p}_i - \mathbf{v}_j\| \tag{4.14}$$

for all $\mathbf{v}_j \in \mathbf{V}^{n\times 2}$ distinct from $\mathbf{p}_{hp,i}$, which is the best approximation to $\mathbf{p}_i$ by elements of $\mathbf{V}^{n\times 2}$. In other words, this projection mechanism is the best one possible using orthogonal projections.

In conclusion, the kind of visualization explained in this section, could well be done without the use of a SOM. One would then do the principle component analysis (PCA) on the samples instead of the SOM weights. Here too, the projected value function could be included in the PCA, to emphasize the visualization of the variance of the value function itself.

However, if we want to include the dynamic clustering as explained at the end of Section 4.3, the SOM is a necessary component in the visualization process. One could

then see the centers of the radial basis functions moving to their respected positions during operation, and see the formation of the value function at the same time.

Due to insufficient data resources of higher dimensions, further research will be necessary when an appropriate MIMO controller becomes available.

CHAPTER 5

THE INVERTED PENDULUM SWING-UP EXPERIMENT

5.1. The inverted pendulum swing-up

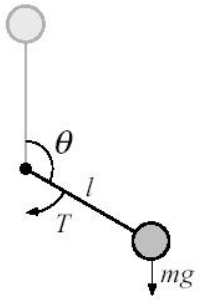

Figure 5.1: Physical Configuration of the Swingup Pendulum Experiment

The physical configuration of the experiment is shown in Figure 5.1, where a mass m is connected to a rigid link of length l. The link is mounted by a joint, that is actuated by a motor with limited torque. Initially, the mass rests in its stable equilibrium, at the bottom position. The goal of the experiment is to get the controller to balance the mass at the top position, starting in the lower equilibrium.

Controlling this one degree of freedom system is non-trivial if the maximal output torque u^{max} is smaller than the maximal load torque mgl. The controller has to swing the pendulum several times to build up momentum and also has to decelerate the pendulum early enough to prevent the pendulum from falling over. Also, there exists no closed-form analytical solution for the optimal solution and complex numerical methods are required to compute it [7].

The nonlinear dynamics are given by:

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{5.1}$$

or

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{1}{ml^2}\left(-\mu\omega + mgl\sin\theta + u\right) \end{bmatrix} \tag{5.2}$$

where $m = l = 1$, $g = 9.8$, $\mu = 0.01$, and $u^{max} = 5.0$.

For a two dimensional state and a one dimensional action, as in equation (5.2), the value function and the system dynamics are represented by a function approximation

network with a two and three dimensional input and a one and two dimensional output, respectively. Normalized radial basis functions are used as the function approximation network (Appendix B).

The value function is written as

$$V(\mathbf{x}(t)) = \sum_{i=1}^{I_v} w_{v,i}\tilde{\Phi}_{v,i}(\mathbf{x}(t)) \tag{5.3}$$

where $I_v = 225$ and represents the 15x15 basis functions of the state space. The value function weights $w_{v,i}$ are initialized as zero. The system dynamics are written as

$$f(\mathbf{x}(t), \mathbf{u}(t)) = \begin{bmatrix} f_1(\mathbf{x}(t), \mathbf{u}(t)) \\ f_2(\mathbf{x}(t), \mathbf{u}(t)) \end{bmatrix} \tag{5.4}$$

$$f_j(\mathbf{x}(t), \mathbf{u}(t)) = \sum_{i=1}^{I_m} w_{m_j,i}\tilde{\Phi}_{m,i}(\mathbf{x}(t), \mathbf{u}(t)) \tag{5.5}$$

where $I_m = 450$ and represents the 15x15x2 basis functions of the state-action space. The system dynamics weights are randomly initialized. Both the value function and the system dynamics basis functions are equally distributed on a rectangular grid. Several simplifications can be made for a rectangular grid. These are described in Section 5.3.

Despite the random initialization of the system dynamics weights, the experiment will have to be deterministic and repeatable. This can be done by taking a constant random seed. To widen our scope, the constant seed is replaced by a time-dependent random seed. Experiments prove that a non-deterministic initialization of the model weights and controller state does not give any problems. The controller handles all cases as it should.

There are three ways of initializing the initial state of the simulated system. First is the deterministic initialization in which the pendulum is returned to its lower equilibrium each time. Second, we could start the simulation at a random angle and zero angular velocity. Third, we could initialize the state in a natural way, by setting the initial state the first way before our simulation begins while subsequent trials are started where the previous trial ended unless the pendulum is over-rotated, when one should delay the next trial until the pendulum has slowed down enough. The third way would be useful when the controller is trained on a the physical setup. We use the first way and set the initial state to a constant value of

$$x(0) = \begin{bmatrix} \theta \\ \omega \end{bmatrix} = \begin{bmatrix} \pi \\ 0 \end{bmatrix}$$

To get our controller working, the exploration factor is required. In the current experiment, the exploration mechanism as described in Section 3.3.3 is used, with the parameters $\tau_n = 0.5$, $\sigma_0 = 0.5$.

The reward signal is given by

$$r(t) = \cos(\theta) \tag{5.6}$$

so that the upper equilibrium has positive reward, while the lower equilibrium has negative reward.

Each trial lasts for 20 seconds unless $|\theta| > 5\pi$ upon which a reward $r(t) = -1$ is given for one second. A trial is regarded successful when $t_{up} > 10s$, where the performance measure t_{up} is increased while $|\theta| < \frac{\pi}{4}$. The pendulum being balanced within 45 degrees from the upper position is considered acceptable in this experiment.

Finding the optimal values of the simulation parameters would require solving the reinforcement learning equations. The values of the simulation parameters used in this experiment are obtained by trial and error, and are given by $\tau_v = 1.0$, $\kappa = 0.1$, $c = 0.1$, $\eta_v = 50$, $\eta_m = 500$, $dt = 0.02$. The experimental results are very sensitive to deviations from these values, and can cause instability of the algorithm, especially concerning the sample rate.

5.2. Experimental results

Figure 5.2 shows the value function obtained from simulation. The initial condition is located in the middle of the figure. Low values of the value function are represented by black, while places of high value are represented by dark grey. When we follow the black line, we see the pendulum building up momentum before it swings to the dark grey area. The pendulum is then being balanced at $\theta = 0$.

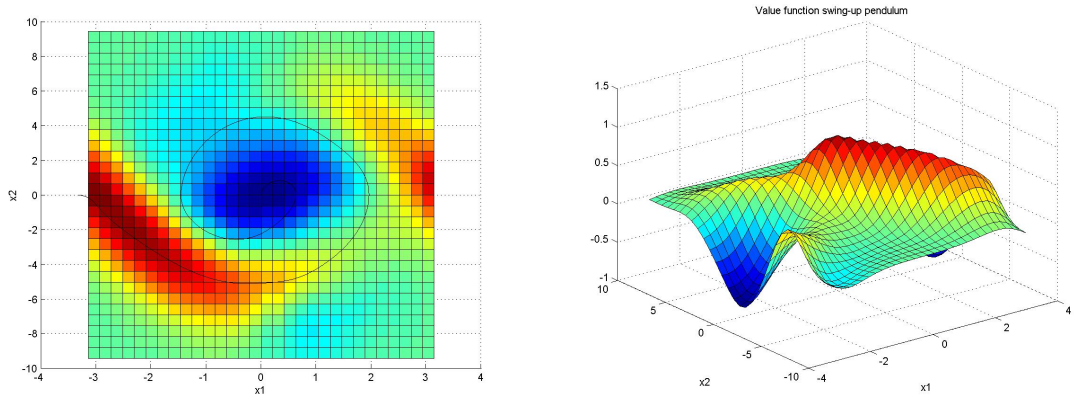

Figure 5.2: A test trial and the value function for the pendulum swing-up task.

Figure 5.3 shows the value function in perspective. Figure 5.3 shows the time in the up-position $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$. Figure 5.4 shows a test trial, where η_v and η_m are set to zero.

The controller in no longer learning. Figure 5.5 shows the control torque applied to the system during the test trial.

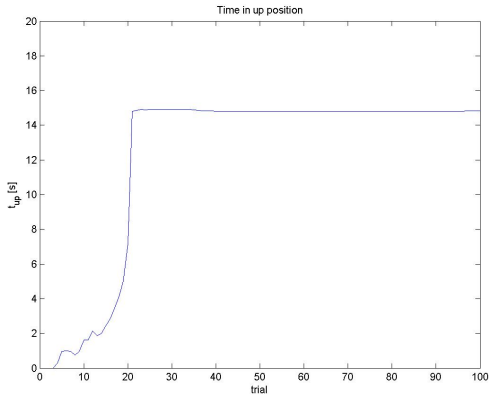

Figure 5.3: The time in up-position

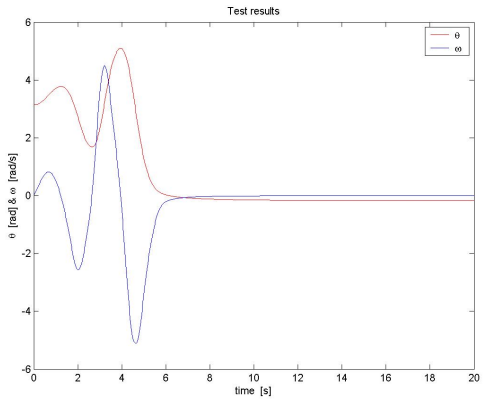

Figure 5.4: A typical test trial

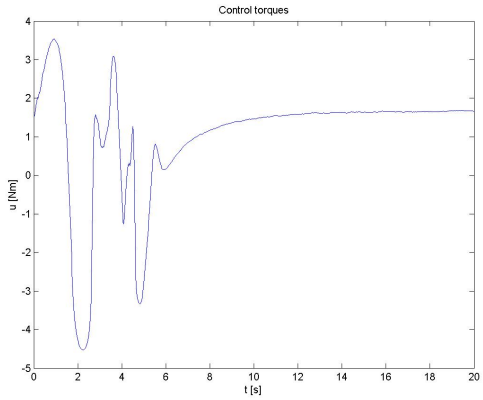

Figure 5.5: The control torque

As can be seen in Figure 5.4, the pendulum is not being balanced at $\theta = 0$ exactly. The angle is a little below zero. Therefore, the control torque in Figure 5.5 is stays at values greater than zero. The reason this occurs is that we have defined a reinforcement signal that is nearly equal for all angles θ in the vicinity of $\theta = 0$. We would have to choose a different reinforcement signal in order to make the angle exactly zero.

5.3. Rectangular grid simplifications

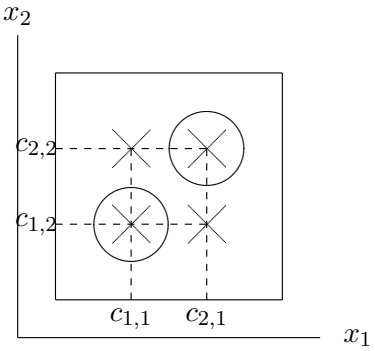

Figure 5.6: Rectangular grid of radial basis functions

Suppose the RBF's are arranged in a rectangular grid. To describe these basis functions four weights and four centers are needed. The centers of the radial basis functions are defined as

$$\mathbf{c}_i = \begin{bmatrix} c_{i,1} \\ c_{i,2} \end{bmatrix} \tag{5.7}$$

for $i \in [1, 4]$. So we have eight center coordinates in total, for a two-dimensional state-space.

As can be seen from Figure 5.6, $\mathbf{c}_3$ and $\mathbf{c}_4$ can be written in terms of $\mathbf{c}_1$ and $\mathbf{c}_2$ and only four center coordinates are needed for the computations. Now, by substitution of equation (B.7), we can write equation (B.10) as

$$\Phi_{v,i}(r_{v,i}) = \exp\left(-\sum_{j=1}^{2} \frac{(x_j - c_{i,j})^2}{2\sigma_j^2}\right) \tag{5.8}$$

$$= \prod_{j=2}^{2} \exp\left(-\frac{(x_j - c_{i,j})^2}{2\sigma_j^2}\right) \tag{5.9}$$

In case of n^2 basis functions, using equation (5.8) we will need to evaluate n^2 exponentials. On the other hand, using equation (5.9) we will only need to evaluate $2n$ exponentials. This is very beneficial for the calculation and memory requirements of both the value function and the system dynamics. Since the basis function representing the system dynamics have the same centers and standard deviations for the first two dimensions, i.e. the system states, there is no need to compute these all over. Equations (5.10-5.11) show us we only have to compute the remaining dimensions of the control output.

$$\Phi_{m,i}(r_{m,i}) = \exp\left(-\frac{(u - c_{i,3})^2}{2\sigma_3^2}\right) \prod_{j=2}^{2} \exp\left(-\frac{(x_j - c_{i,j})^2}{2\sigma_j^2}\right) \tag{5.10}$$

$$= \exp\left(-\frac{(u - c_{i,3})^2}{2\sigma_3^2}\right) \Phi_{v,i}(r_{v,i}) \tag{5.11}$$

By these simplifications, all computational redundancy is eliminated. When we use a multi-layer perceptron, or other techniques such as dynamic clustering of the radial basis functions, this redundancy does no longer exists. This has to be accounted for in finding a optimal method of function approximation.

5.4. An algorithm for circular state-spaces

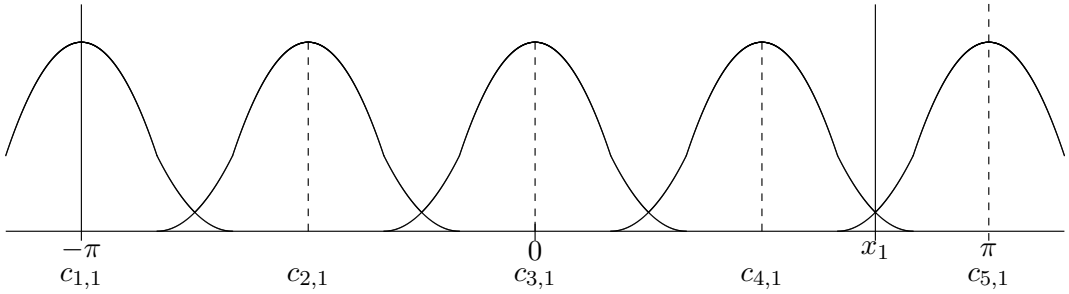

Figure 5.7: Circular state-space for $I = 5$ radial basis functions

Because we are working with a cylindrical state-space, a couple of serious problems arise. For given $x_1 \in [-\pi, \pi)$ and $c_{i,1} \in [-\pi, \pi)$ it is seen that

$$\|x_1 - c_{i,1}\| \in [0, 2\pi) \tag{5.12}$$

To clarify the problem, we consider the following cases

$$\begin{array}{lllll} 1) & x_1 < 0 & \wedge & c_{i,1} < 0 \\ 2) & x_1 < 0 & \wedge & c_{i,1} > 0 \\ 3) & x_1 > 0 & \wedge & c_{i,1} < 0 \\ 4) & x_1 > 0 & \wedge & c_{i,1} > 0 \end{array}$$

From these cases, the following table is constructed

	$x_1 - c_{i,1} \in$	$\|x_1 - c_{i,1}\| \in$	$2(x_1 - c_{i,1} > 0) - 1$	$\|x_1 - c_{i,1}\| > 2\pi - \|x_1 - c_{i,1}\|$
1)	$[-\pi, \pi)$	$[0, \pi)$	± 1	false
2)	$[-2\pi, 0)$	$[0, 2\pi)$	-1	true/false
3)	$[0, 2\pi)$	$[0, 2\pi)$	$+1$	true/false
4)	$[-\pi, \pi)$	$[0, \pi)$	± 1	false

Table 5.1: Data needed for deriving cylindrical state-space algorithm

As can be seen from the last column of Table 5.1, possible overflows occur in case (2) and (3). That is, the distance $\|x_1 - c_{i,1}\|$ is not the shortest distance. When calculating

$\Phi_{i,1}$ with the shortest distance instead, the partial derivative $\Phi'_{i,1}$ changes accordingly. The mapping of x_1 onto $\Phi_{i,1}$ and $\Phi'_{i,1}$ as in equation (B.7) and (B.17) thus becomes non-trivial.

Depending on whether an overflow occurred, and whether it was on the right or on the left of the center, the sign of the derivative can be determined. The algorithm thus becomes

foreach $i \in [1..I]$
  $dx = x_1 - c_{i,1}$
  $absdx = \|dx\|$
  $signdx = 2(dx > 0) - 1$; *are we on the right?*
  $of = 2 * (dxabs > 2\pi - dxabs) - 1$; *was there an overflow?*
  $absdx = \min(absdx, 2\pi - absdx)$
  $\Phi_{i,1} = \exp(-\frac{absdx^2}{\sigma_1^2})$
  $\Phi'_{i,1} = -signdx * of * \frac{dx}{\sigma_1^2} * \Phi_{i,1}$
end

CHAPTER 6

THE ONBOARD RENDEZVOUS CONTROL SYSTEM

6.1. Tasks and functions

The intention of this chapter is to provide the reader with a short overview of the typical tasks, functions and system hierarchy of an automatic onboard control system for a rendezvous and docking mission, without entering into details of the actual design. For more information on the actual docking process, refer to [8]. The figures in this chapter are mostly copied from [3].

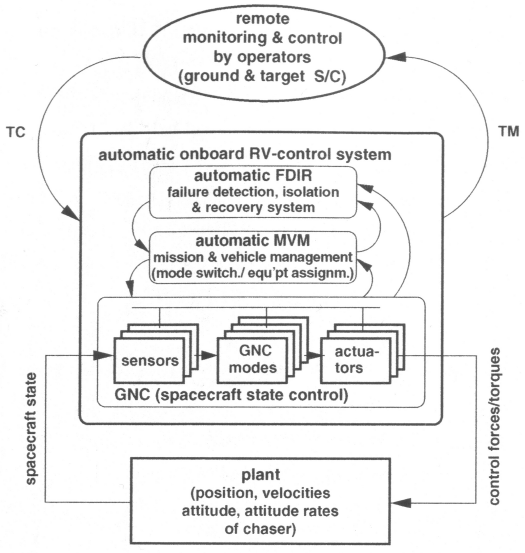

Figure 6.1: Hierarchy of the control system for RVD

During the rendezvous and docking (RVD) process, the automatic onboard system has to fulfill the following tasks:

- preparation and execution of manoeuvres and continuous control of trajectory and attitude (guidance, navigation and control (GNC))
- sequencing of phases, GNC modes or manoeuvres, and scheduling of equipment for such modes (mission and vehicle management (MVM))
- detection and recovery from system and equipment failures and from critical state vector deviations (failure detection, isolation and recovery (FDIR))

- data exchange concerning the rendezvous process and the onboard control system with the ground control center and the target space station

To fulfill all the above listed tasks, the onboard control system for RVD will have to be designed according to a hierarchical structure. The typical hierarchy of the overall control setup for automatic rendezvous is shown in Figure 6.1. This simplified figure shows only the levels of authority, not the actual functional relations within such a system. For instance, there will be the need to have failure detection functions at all levels, including the level of the GNC software functions and in the sensor and actuator hardware.

6.2. Guidance, navigation and control

The control loops for attitude and trajectory control are contained within the guidance, navigation and control (GNC) functionality, that is implemented in software in the onboard computer. This includes the truster management system (TMS). A block diagram of the control loop is shown in Figure 6.2.

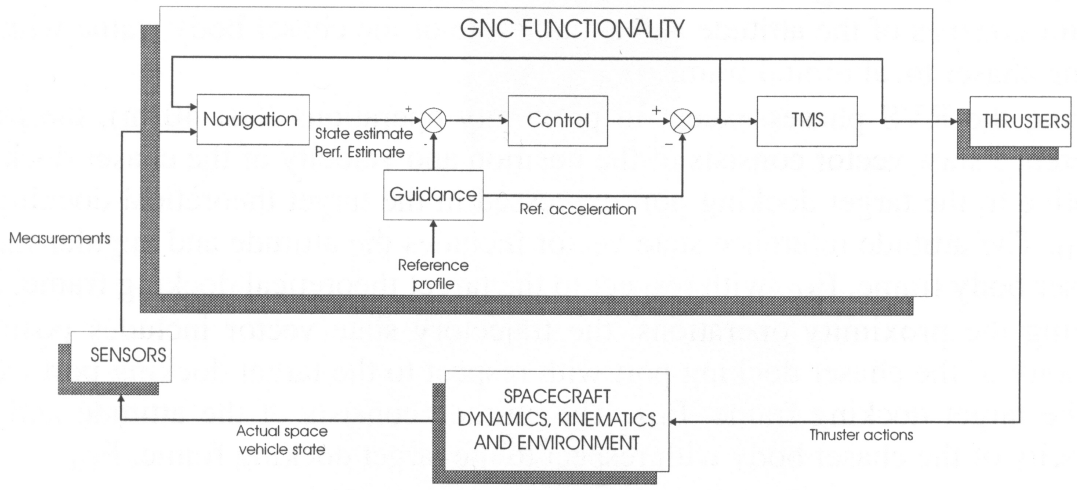

Figure 6.2: GNC functions

Depending on the distance from the target vehicle, various translation and attitude manoeuvres have to be controlled. This requires a reconfiguration of the control loops. The proper algorithm and parameters, i.e., the GNC mode, have to be selected for each phase of the approach.

As long as the distance between the two vehicles is large enough, each degree of freedom (DOF) of rotation and translation may be controller independently by a SISO controller. During the last part of the approach, when the docking mechanism of the chaser

vehicle has to be aligned with the docking port of the target vehicle, all motions are coupled (see Section 6.3). In this case, a MIMO control system will have advantages.

6.2.1. The navigation function

The task of the navigation function is to provide the controller and the guidance function with the necessary information on the present state of the vehicle. For each navigation mode, the navigation function consists of a Kalman filter, which processes the information of attitude and trajectory sensors and propagates the vehicle state vector by using knowledge of the dynamic behavior and information on the actual thrust commands. Its purpose is to obtain an estimation of the state with reduced noise errors. Such a filter will also be helpful in cases where the sensor information is only intermittently available.

6.2.2. The guidance function

The guidance function defines the set values for nominal evolution of the spacecraft state, i.e. the references for the control of position, velocities, attitude and angular rates at each point in time. For example, the propagation of the instantaneous position of the center of mass in the vehicle body frame according to the propellant consumption during the mission. These set values will then be compared with the estimated actual values, provided by the navigation function, enabling the control function to prepare the control commands.

6.2.3. The control function

The task of the control function is to provide the force and torque commands necessary to correct the deviations of the actual attitude and position from the nominal ones and to ensure stability of the vehicle. While the guidance function provides the nominal or reference state, and the navigation function estimates the actual state, from the difference of the two states the control function produces actuation commands to compensate for the effects of disturbances and errors.

6.2.4. The thruster management system

The truster management function transforms the torque and force commands into 'on/off' commands for the individual thrusters. This functions is of particular importance for vehicles which have their thrusters located in an unbalanced arrangement w.r.t. the center of mass. In such cases each translation force and each rotation torque has to be produced by a combination of various thrusters with burns of different duration.

6.3. The spacecraft dynamics, kinematics and environment

The *plant* in Figure 6.2 is the block representing the six degrees of freedom motion dynamics of the spacecraft, i.e. the dynamics of translational motion (position dynamics) and rotational motion (attitude dynamics). The coupling between translational and

rotational motion is due to the orbital rotation (small) and the translational components $r\sin(\alpha)$ that results from a rotation angle α at a distance r from the center of rotation. The latter effect plays a role when the vehicle is controlled w.r.t. a frame which does not originate in its COM. This is the case when both lateral and rotational alignment with the target docking axis has to be achieved prior to contact. In all other cases, coupling of translational and rotational motions is rather small, and can usually be neglected.

6.4. The proximity operations

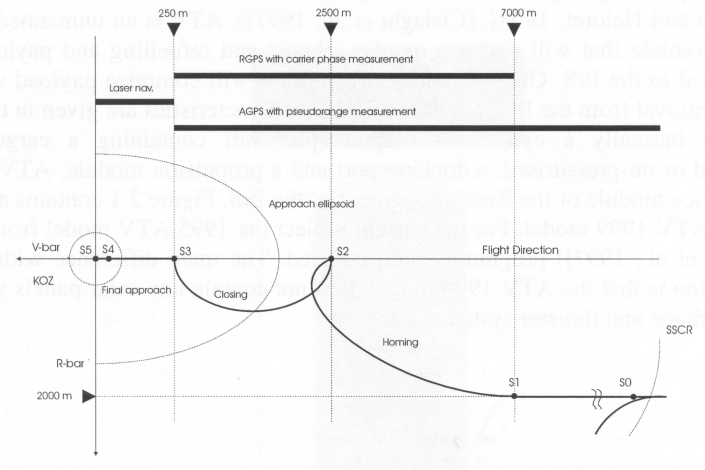

Figure 6.3: Approach strategy to docking port

Figure 6.3 illustrates the approach strategy of the European ATV to the ISS. From the point labeled (S2), approach into the *approach ellipsoid* (AE) can commence only after permission from the ISS. Under RGPS navigation, a radial boost transfer manoeuvre leads the ATV into the AE to a point (S3) where acquisition of the target reflectors by the optical rendezvous sensor can take place.

Under optical RVS navigation, the straight line forced motion from the point labeled (S3) to the point labeled (S4), leads to the docking port. The major disadvantage of the straight line forced motion approach for closing is the high propellant consumption, however, approach recovery is easier than for tangential transfer. Also, it is more suitable for entry of the *approach corridor*, a narrow cone shaped corridor of $\pm$ 10 degrees, defined by the target station.

The velocity profile along the approach line consists of several distinct phases of acceleration, deceleration and constant velocity, before contact with the target interfaces is established. The results presented in the next chapter do only partly represent the real scenario. During simulation, the chaser is held at a fixed distance of 40 meters with a constant accuracy requirement in position and velocity. A realistic scenario would include

an increase in accuracy requirements in position and velocity, due to the approach corridor and the angular operating range of the optical sensors.

The sensors used during this phase are a scanning laser range finder (< 500 m), and a camera (< 20 m). These sensors measure the relative position and relative attitude. The required velocity and angular rate measurements are obtained by differentiation w.r.t. time, of the relative position and relative attitude measurements.

6.5. Controller design for the proximity operations

In the last part of the approach, an important requirement in the controller design is that the chaser must be able to follow the motion of the docking port of the target. In the case of target thruster firings and of motions due to structural flexibility of the target during the final approach of the chaser, the frequency content of these motions may lead to higher bandwidth requirements than in the other approach phases.

Although steady state errors of position and angular alignment may fit into the reception range of the docking mechanism, transient errors may be large. The transient response of the controller must be such that, the sum of the instantaneous values of lateral position and angular misalignment between the docking interfaces of the chaser and target will be smaller than the reception range, denoted as the maximum distance in Figure 6.4. Generally, in the case of docking, the keep the impact low, contact and lateral velocities and angular rates should be as low as possible.

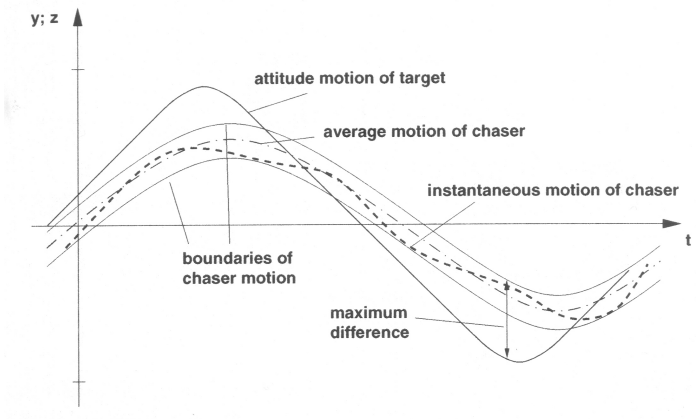

Figure 6.4: Motion of target and chaser at docking

Both the relative lateral position and the relative attitude have to be controlled simultaneously to achieve full translational and rotational alignment of the docking interfaces. This needs, in addition to the lateral position, an onboard estimation of the relative orientation of the docking axis of chaser and target. A consequence of this scheme is that

rotational and translational motions are now coupled, which has important consequences for the controller, as it requires more advanced controller design techniques.

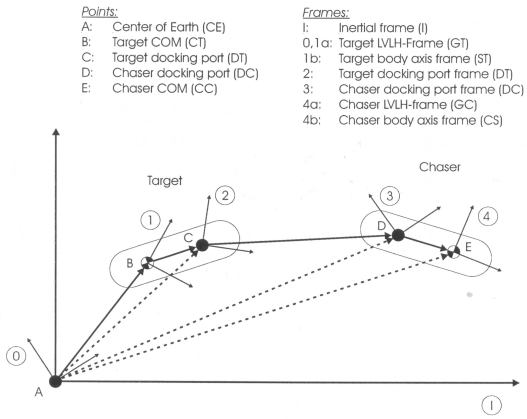

Figure 6.5: Schematic overview port-to-port docking problem

A schematic overview of the relative port-to-port docking problem is given in Figure 6.5. In order to avoid problems because of the coupling between the relative port-to-port translational and rotational motions, the concept of *R-control* is introduced [3], it uses the distance between the chaser's COM and a point at which the chaser's COM would be in case of perfect docking, as depicted in Figure 6.6. This concept decouples the relative translational and rotational motions, thereby making it possible to use the same controllers in the RVS phase as in the GPS phase.

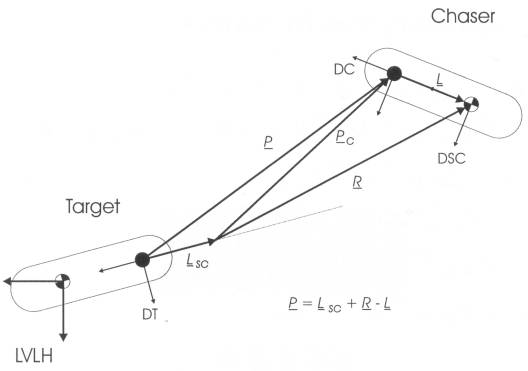

Figure 6.6: R-transform

When the relative translational and rotational motion are decoupled, it is possible to design an array of six independent controllers for the resulting six degrees of freedom. The array consists of four PD controllers and two controllers based on reinforcement learning that control the lateral port-to-port positions and velocities, that is, the position control in the plane perpendicular to the docking axis.

6.6. ISS disturbance motion modeling

The structural flexibility of the target, is modeled as a periodic high frequency disturbance angular motion round the COM. Since the docking port is at a certain distance from the COM, these angular motions translate into lateral (saw-tooth) motions of the docking port, represented by the solid line in Figure 6.4. The rounded-off corners are due to the available thrust level and the inertia of the target spacecraft.

Within the simulator, the saw-tooth disturbance is represented by a Fourier series. In the current experiment, the first ten harmonics are used. Further disturbances on the ATV and the ISS, as described in Section 1.2, are not taken into account. Because the scope of the project was emphasized on making an evaluation of the control techniques, the same conditions hold as in the original experiments.

CHAPTER 7

EXPERIMENTAL RESULTS

The chaser is being held at a fixed relative position, 40m behind the ISS. The array of six independent controllers control the lateral port-to-port positions and velocities, that is, the control of the position and the velocity in the plane perpendicular to the docking axis. The array consists of four PD controllers and two controllers based on reinforcement learning.

The reinforcement learning controllers represent a non-linear mapping of error and error rate to the controller output

$$u(t) = f(e_{rtrans}(t), \dot{e}_{rtrans}(t), e_{p2p}(t), \dot{e}_{p2p}(t)) \tag{7.1}$$

where

$$e_{rtrans}(t) = x_{gui}(t) - R(x_{nav})(t) \tag{7.2}$$

represents the error that is controlled to zero, $R(\cdot)$ denotes the R-transform (Section 6.5), and

$$e_{p2p}(t) = x_{gui}(t) - x_{nav}(t) \tag{7.3}$$

represents the port-to-port error that forms the input of the reinforcement signal. Since the chaser is being held at a constant position, $x_{gui}(t)$ can be considered constant.

The goal of the controllers is to keep the both errors and error-rates, within certain predefined boundaries, given limited thrust capabilities. One controller for each direction in the plane perpendicular to the docking axis. Usually, these boundaries are given by the approach corridor and the reception range of the target vehicle.

The signal constraints are given by $e_{rtrans}(t), e_{p2p}(t) < 0.3m$, $\dot{e}_{rtrans}(t), \dot{e}_{p2p}(t) < 0.03m/s$ and $u_{max} = 150N$.

7.1. Results of the original controllers

First of all, the simulation results obtained with the use of the the two original reinforcement learning controllers will be given.

The original controllers used to control the lateral relative port-to-port motion were implemented as n-step Temporal Difference controllers. Function approximation was done using by means of fuzzy logic. The definition of fuzzy regions for the state-action space is a logical extension of the 'crisp' cells in the lookup-table. An important advantage of such

fuzzy systems, is that the results can be interpreted in terms of *if-then* rules and possibly validated with expert knowledge.

The controllers were initialized as linear PD controllers of the same configuration as used for all other DOF's. A simulation lasts for 10000 s. These are the result to which we compare the results obtained using the techniques explained in this thesis. Only the last 400 s are depicted below.

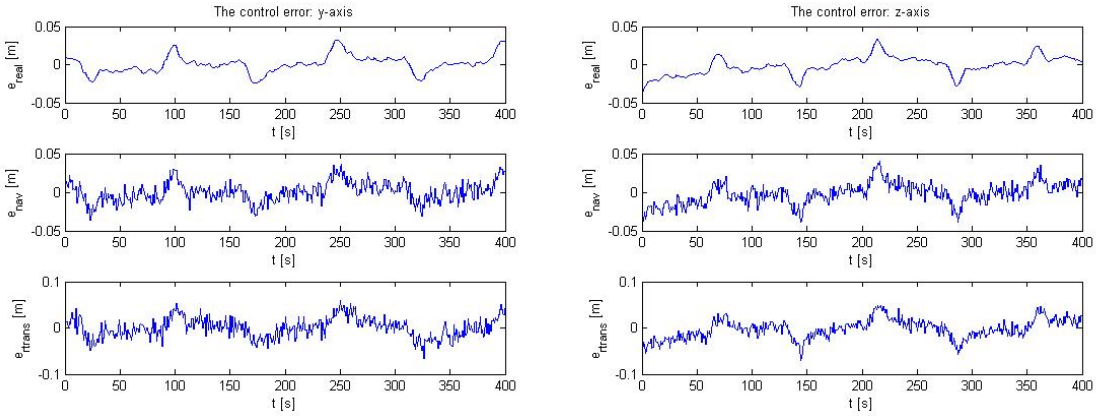

Figure 7.1: Translational position errors (original experiment)

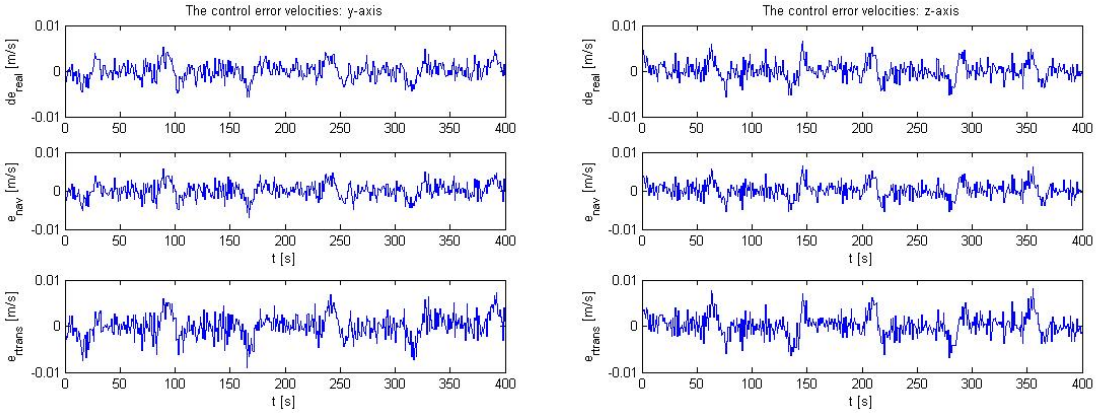

Figure 7.2: Translational velocity errors (original experiment)

Figure 7.1 shows the control errors, $e_{real}(t)$, $e_{nav}(t)$ and $e_{rtrans}(t)$ for both axes. Figure 7.2 shows the control error velocities $\dot{e}_{real}(t)$, $\dot{e}_{nav}(t)$ and $\dot{e}_{rtrans}(t)$ for both axes. Figure 7.3 shows the control torques generated by the controllers. The controller is performing well within the given constraints, while following the oscillation caused by the flexibility of the target.

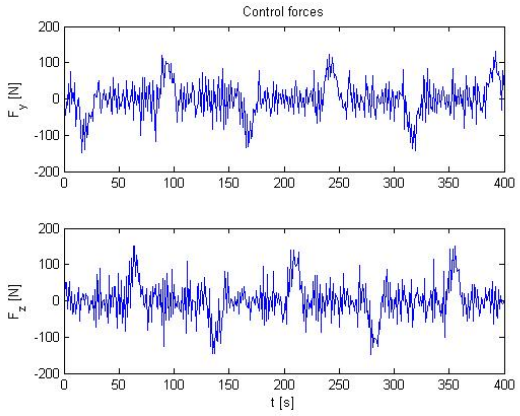

Figure 7.3: Control forces (original experiment)

7.2. Results of the continuous space and time reinforcement learning controllers

The representation of the value function and the system dynamics conform with the definition in Section 5.1. In transferring the controller to the parallel configuration needed for the control of the lateral relative positions, some minor modifications to the controller will take care of the process differences such as immediate reward, trial termination, signal bounds on the process state and radial basis function handling.

The signal limits are the same as in the original experiment and are given by $e_{rtrans}(t), e_{p2p}(t) < 0.3m$ and $\dot{e}_{rtrans}(t), \dot{e}_{p2p}(t) < 0.03m/s$. The simulation parameters remain unchanged with respect to the pendulum swing-up, that is $\tau_v = 1.0$, $\kappa = 0.1$, $c = 0.1$, $\eta_v = 50$, $\eta_m = 500$, $dt = 0.02$. A trial is considered successful when it remains at a fixed distance of 40 meters for 400s without exceeding the signal bounds.

The reward construction used is this chapter is based on the following assumption. Through the definition of the reward function, the controller will try to minimize the control error and control error velocities. This means, that if we depict our state space we want the controller to be in the second and fourth quadrant. When the error is positive, the error rate should be negative. When the error is negative, the error rate should be positive.

Experiments were done using three different reward functions:

1. The original implementation of the reward function was linear (Figure 7.4). It is described by

$$r(e_{p2p}, \dot{e}_{p2p}) = -\frac{1}{0.05}|e_{p2p} + 10 * \dot{e}_{p2p}| \tag{7.4}$$

As can be seen, the controller is forced to stay in the second or fourth quadrant.

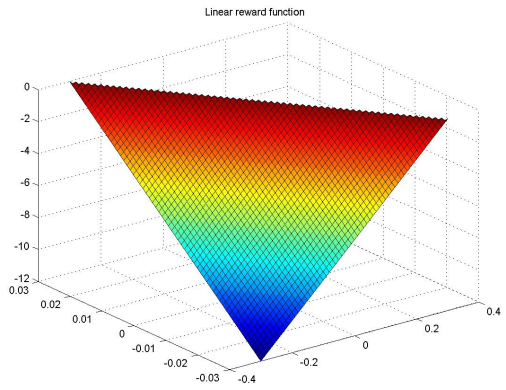

Figure 7.4: reward function in the linear regime

Furthermore, there is no stimulant for the controller to go to the origin of the state space. When a trial terminates prematurely, that is if one of the states exceeds the signal bounds, a reward of -15 is given.

2. The origin attraction is resolved by the following function

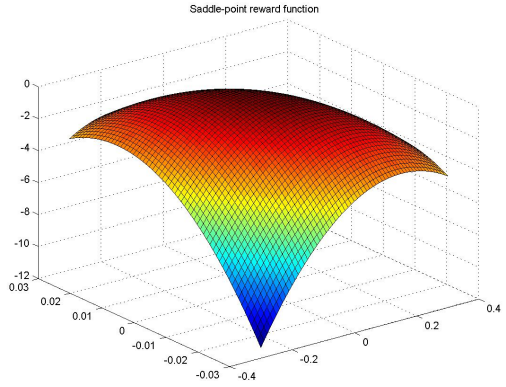

Figure 7.5: Reward function based on saddle-point

$$r(e_{p2p}, \dot{e}_{p2p}) = -42 * (e^2_{p2p} + e_{p2p} * 10 * \dot{e}_{p2p} + 100 * \dot{e}^2_{p2p}); \tag{7.5}$$

As can be seen, the controller is still forced to the second and fourth quadrant, but at the same time, the origin is given priority. On premature trial termination, a reward of $r = -25$ is given.

3. The origin can be further accented by square-rooting the saddle-point function

$$r(e_{p2p}, \dot{e}_{p2p}) = -75 * \sqrt{e_{p2p}^2 + e_{p2p} * 15 * \dot{e}_{p2p} + 100 * \dot{e}_{p2p}^2}; \tag{7.6}$$

Using this reward function, however, did not make the controller perform better.

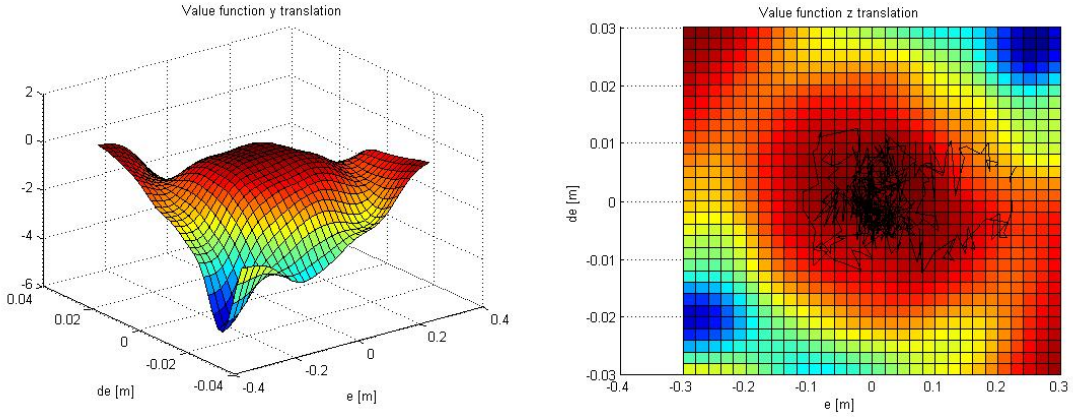

Figure 7.6: Value function and last successful trial for y-axis controller

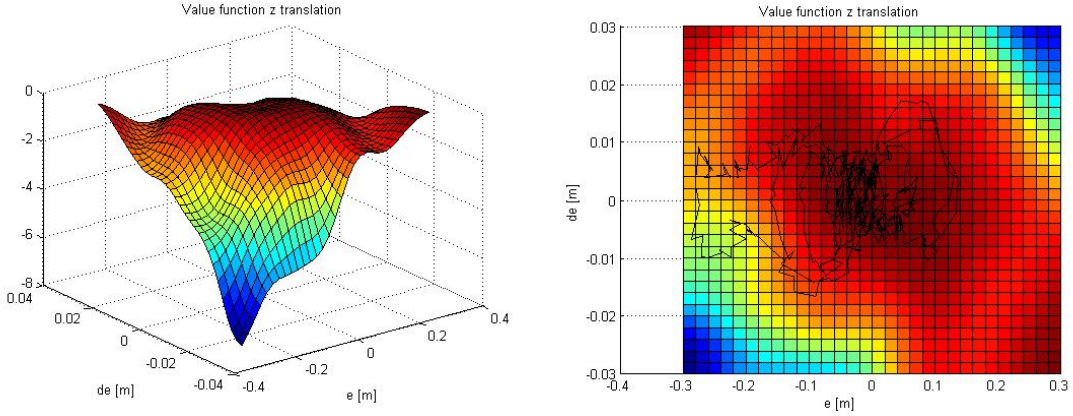

Figure 7.7: Value function for z-axis controller

First of all, the algorithm clearly succeeds in fulfilling the objective. The chaser matches the disturbance of the target docking port within the given signal bounds, and therefore within the reception range of the target. It is however, not as stable as in the pendulum swing-up experiment.

Figure 7.6 - 7.11 show the results obtained during simulation using the continuous time and space reinforcement learning controllers. When we compare Figures 7.6 and 7.7 to the value function obtained in the pendulum swing-up experiment, displays a vastly more

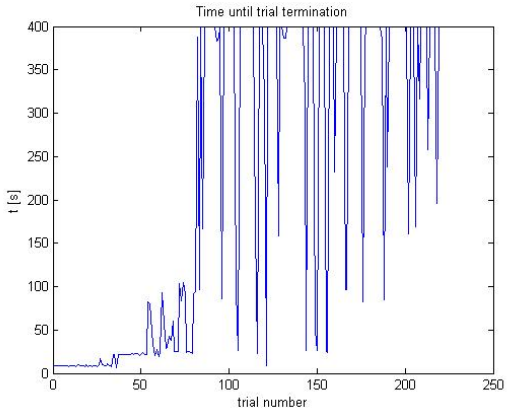

Figure 7.8: Time until trial termination

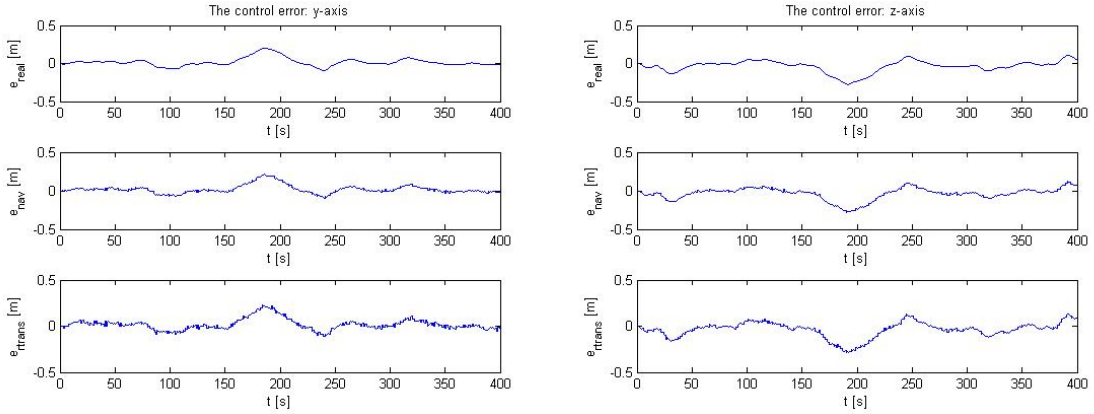

Figure 7.9: Translational position errors

complex scene, from which less information can be extracted than from the value function of the pendulum swing-up experiment. Also, Figures 7.6 and 7.7 show a much more erratic trial than in the pendulum swing-up experiment. Since the ATV-ISS docking mission is more complex situation, this is not completely unexpected.

Compared to the results of the original experiment, Figures 7.9 and 7.10 show a much bigger error and error rate. Possible causes include the difference in initialization. While the original controller was initialized as a linear PD controller, the current controller is initialized randomly (Section 7.3). This has a certain influence on the convergence of the algorithm. Another reason the performance is worse, is the information content of the function approximation method, as will be explained in the following section.

The control forces shown in Figure 7.11 are smoother and less noisy compared to Figure 7.3. This seems to be the case for the error and error rate figure too, and could

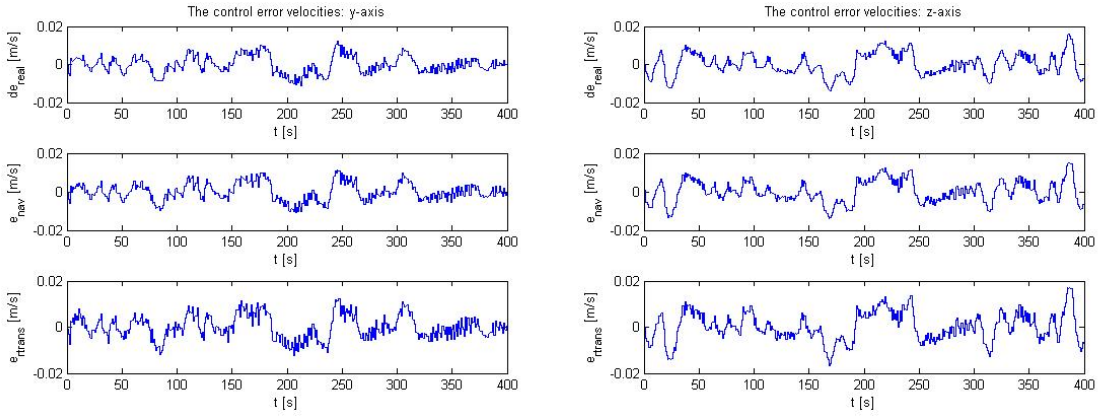

Figure 7.10: Translational velocity errors

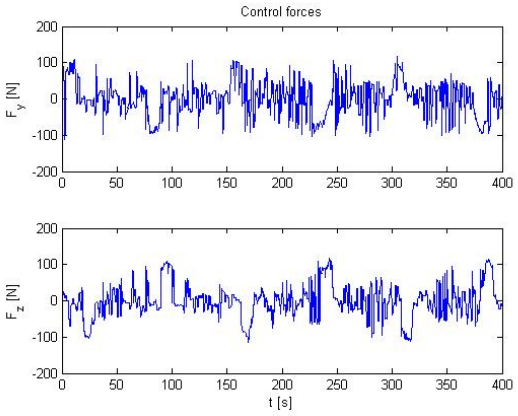

Figure 7.11: Control Forces

indicate a difference in exploration.

In conclusion, due to many factors, the algorithms remain difficult to compare. Although the mathematics of the current algorithm are more advanced, this does not show in the results.

7.3. Improving controller performance

In order to improve the controller's performance, there are a number of options. A discussion of some simple tests and some more elaborate methods will follow.

First, simply increasing the number of trials had no observable influence on the accuracy of the controller. After some time of training, the controller does not improve any more.

Second, the gradually reducing the exploration noise has been tried. The total amount of elapsed time, or number of trial has been taken into account linearly, to ensure that there is no significant exploration at the end of the experiment. There was indeed a definite improvement in the stability of the trail duration.

Third, dynamically changing the maximum and minimum levels of expected reward $V_{0,y}$, $V_{1,y}$, $V_{0,z}$, and $V_{1,z}$ (equation (3.42)) after each trial seems to deteriorate the learning performance. The accuracy is improved, but overall stability is degraded.

Fourth, to further improve performance, the policy could be slightly modified by removing the arctan. Instead, actions could be rewarded or punished through the reward function. This has not been implemented yet.

And for the more elaborate methods:

The information a function approximation method of the kind used in this experiment, that is, the use of normalized radial basis functions, is limited for a number of reasons. First, the general amount of information the function approximation method can hold, is limited per definition. Second, the rectangular distribution of the radial basis functions, is far from optimal. In function approximation, some regions are more important than others in their representation. Clustering schemes such as the self-organizing map could be used to improve on this second limitation, however, the simplifications that can be made using a rectangular grid (Section 5.3) will be lost.

In the current experiment, the sample rates used were not uniform. The controller had a sample rate of 0.02s, while the rest of the system was sampled at 1s. This unfortunate situation came forth from the combination of the controller used on the swing-up pendulum experiment, and the ATV-ISS simulator as used in its original form. Trying to change the sample rate of the controller, appears to lead to instability in the Euler integration of the value function and system dynamics weights, used to represent them, while changing the sample rate of the rest of the system, so that continuous measurement data and actuation capabilities become available, leads to an unstable algorithm for unknown reasons, even when the measurement noise is neglected. A possible cause could be the limited information capabilities of the function approximation method. An example of such an unstable algorithm is depicted in Figure 7.12.

Simulation shows us that the removal of the measurement noise does not result in better control performance. One would expect exactly the opposite to be true, although satisfactory results are obtained in neither case. A probable reason for this behavior is that the measurement noise could be acting as some sort of high-frequency exploration in addition to the controller's exploration noise. Simulation with measurement noise produce the highest peaks in trial duration, while simulation without measurement noise results in

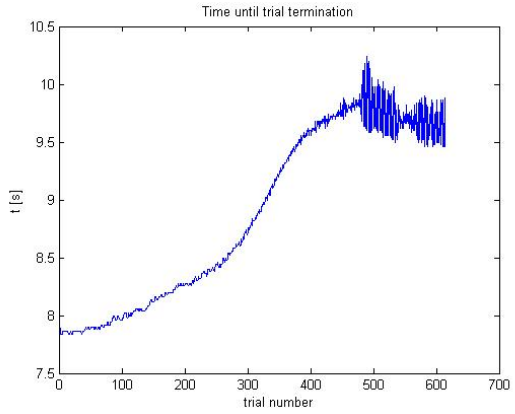

Figure 7.12: Example of instability of the algorithm

much more stable learning.

A possible solution to the problem of the unstable Euler integration is to replace the discrete Euler integration method with continuous time integrators. This should have eliminated the problem of unstable integration, but in trying to do so, the computational environment (Matlab), persistently aborts with a segmentation fault, or the message of singularities within the algorithm, for both fixed-step and variable-step integration. For the moment, it is unclear why the environment became unstable.

In order to solve the problem of limited information capabilities of the normalized radial basis functions, the choice of a more suitable function approximation method, such as the multi-layer perceptron, would be the next step in improving overall capabilities of the algorithm. However, it is the authors opinion, that such a decision, would only make sense, when the issue of the unstable Euler integration method is resolved.

Since the use of the multi-layer perceptron is expected to give rigorous improvement in performance of the algorithm (Appendix C), however, it is advisable to explore the full potential of the MLP at a later stage. It must be accounted for, that the use of the MLP will confront the researcher with some unexpected matters concerning the computational demands. First of all, again, simplifications involving the use of rectangular grid (Section 5.3) will not hold. Second, the convergence properties of the MLP are much worse. That is, the computation time of a single trial will take less time, because the MLP does not suffer from the curse of dimensionality, but overall computation will almost certainly consume a multitude of the current simulation time.

Again, the first attempt to integrate the use of an MLP, without the use of continuous time integrators, were unsuccessful. The structural design of such a controller, is very complex, so a minor probably could have caused this. Unfortunately, there was not enough

time, to search for them.

Furthermore, the reinforcement learning controllers used in the original setup, were initialized as linear PD controllers of the same configuration as used for all other DOF's, and form a non-linear extension to PD-controllers, while the controllers used in the current experiment were initialized randomly. While it is difficult to transfer this condition to the current controllers, because of the radial basis functions, it should be possible to reshape the policy (equation (3.20)) so that it starts off as a PD controller. For example, we could leave out the sigmoid function and initialize the partial derivative of the system dynamics as a constant, e.g. 1, while initializing the partial derivative of the value function as a PD linear control surface.

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS

8.1. Conclusion

In this report, an attempt to improve the current control strategies used in the control of the relative lateral port-to-port motion during the proximity operations of the rendezvous and docking mission of the ATV to the ISS has been described. To that end, advanced reinforcement learning techniques in continuous space and time have been applied.

The existing algorithm was improved, by slightly modifying the original equations. After testing the algorithm on the inverted pendulum swing-up, it was ported to the ATV-ISS simulator without any trouble.

A severe drawback of the reinforcement learning algorithm, is that it is not particularly suitable for mission-critical tasks, such as the docking procedure, because it learns on the basis of trail-and-error and is therefor not hundred percent reliable, even when the exploration is disabled and the learning parameters are set to zero.

Furthermore, there are some minor inconveniences regarding the mathematical constraints of the algorithm. For example, the system dynamics $f(\mathbf{x}, \mathbf{u})$ have to be linear and $r(\mathbf{x}, \mathbf{u})$ has to convex with respect $\mathbf{u}$, respectively. For most systems, however, this is not a real problem.

The greatest advantage of reinforcement learning, is that it is able to compensate for incomplete modeling of the system dynamics, wearing, and environmental fluctuations, and acquires an optimal controller for an optimal model.

Furthermore, an extension to the self-organizing map has been proposed as the technology that enables us to dynamically cluster a high-dimensional state space, as a means to create a suitable distribution of the radial basis functions, while retaining the possibility to extract valuable visual information and to monitoring the algorithm during operation. This enables us to see inside a continuous space and time reinforcement learning controller and learn from its evolution.

8.2. Recommendations

Further research in this area is certainly necessary, therefore many improvements have been suggested in Section 7.3. The most important ones being the use of the continuous equations, and the use of the multi-layer perceptron.

When all the issues stated in Section 7.3 are resolved, the design of a MIMO controller should be considered. While the design of a MIMO controller using the current control strategy will not be advantageous, due to the the curse of dimensionality, it could well be that in applying the above improvements, the controller will perform significantly better and will demand much less computational resources.

Further, both the exploration signal and the immediate reward play a significant role in the behavior of the reinforcement learning algorithm. Slightly modified extremes of the reward function, for example, may lead to a collapse of the learning curve. Therefore, further research into making these processes adaptive could lead to a significant improvement in the overall performance of the reinforcement learning algorithm.

Also, it would be good to investigate the use of adaptive learning rates to refine the learning as specified by some objective when needed. At moments the controller succeeds in controlling the task at hand, a fine-tuning phase could further improve the controller performance.

Genetic algorithms could be used to explore the parameter space for a global optimum, although this process would, given the current computational resources, take too much time to be of any use.

APPENDIX A

THE HJB EQUATION FOR DISCOUNTED REWARDS

We devide the integral (3.5) into two parts $[t, t+\Delta t]$ and $[t+\Delta t, \infty]$ and then solve the optimization problem as

$$V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(s)\in\mathbf{U}, t\le s<t+\Delta t} \left[\int_t^{t+\Delta t} e^{-\frac{s-t}{\tau_v}} r(s)ds + e^{-\frac{\Delta t}{\tau_v}} V^*(\mathbf{x}(t+\Delta t)) \right] \tag{A.1}$$

For small Δt, the first term is approximated as

$$r(t)\Delta t + \mathcal{O}(\Delta t) \tag{A.2}$$

and the second term is Taylor expanded as

$$V^*(\mathbf{x}(t+\Delta t)) = V^*(\mathbf{x}(t)) + \frac{\partial V^*}{\partial \mathbf{x}(t)} f(\mathbf{x}(t), \mathbf{u}(t))\Delta t + \mathcal{O}(\Delta t) \tag{A.3}$$

By substituting (A.2) and (A.3) in (A.1) and collecting $V^*(\mathbf{x}(t))$ on the left-hand side, we have an optimality condition for $[t, t+\Delta t]$ as

$$(1-e^{-\frac{\Delta t}{\tau_v}})V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(s)\in\mathbf{U}, t\le s<t+\Delta t} \left[r(t)\Delta t + e^{-\frac{\Delta t}{\tau_v}} \frac{\partial V^*}{\partial \mathbf{x}(t)} f(\mathbf{x}(t), \mathbf{u}(t))\Delta t + \mathcal{O}(\Delta t) \right] \tag{A.4}$$

By dividing both sides by Δt and taking Δt to zero, we have a condition for the optimal value function

$$\frac{1}{\tau_v} V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(t)\in\mathbf{U}} \left[r(\mathbf{x}(t), \mathbf{u}(t)) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} f(\mathbf{x}(t), \mathbf{u}(t)) \right] \tag{A.5}$$

APPENDIX B

NORMALIZED RADIAL BASIS FUNCTIONS

A normalized radial basis function network consists of three types of parameters:

- The *output layer weights* which determine the heights of the basis functions.
- The *centers* which determine the positions of the basis functions.
- The *standard deviations* which determine the form of the radial basis functions.

Let

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \in \mathbf{R}^n \tag{B.1}$$

be in input vector to the NRBF network. Then, the Mahalonobis norm r_i is computed as

$$r_i = \|\mathbf{x} - \mathbf{c}_i\|_{\mathbf{\Sigma}_i} \tag{B.2}$$

where the center vector $\mathbf{c}_i$ is defined as

$$\mathbf{c}_i = \begin{bmatrix} c_{i,1} & c_{i,2} & \cdots & c_{i,n} \end{bmatrix}^T \in \mathbf{R}^n \tag{B.3}$$

and the norm matrix $\mathbf{\Sigma}_i$ as

$$\mathbf{\Sigma}_i = \begin{bmatrix} \sigma_{i,1}^{-2} & 0 & \cdots & 0 \\ 0 & \sigma_{i,2}^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{i,n}^{-2} \end{bmatrix} \in \mathbf{R}^{n^2} \tag{B.4}$$

For diagonal $\mathbf{\Sigma}_i$, identical elements lead to a true *radial* basis function while different elements lead to a symmetric basis functions with elliptic countours. Off-diagonal elements allows for rotations of the basis functions.

Let

$$\mathbf{\Sigma}_i = \mathbf{\Sigma} \tag{B.5}$$

be a diagonal norm matrix with identical elements. We write equation (B.2) as

$$\|\mathbf{x} - \mathbf{c}_i\|_{\mathbf{\Sigma}_i}^2 = (\mathbf{x} - \mathbf{c}_i)^T \mathbf{\Sigma}_i (\mathbf{x} - \mathbf{c}_i) \tag{B.6}$$

$$= \sum_{j=1}^{n} \frac{(x_j - c_{i,j})^2}{\sigma_j^2} \tag{B.7}$$

The normalized radial basis function (NRBF) network output is calculated by

$$\hat{y} = \sum_{i=1}^{I} w_i \tilde{\Phi}_i(\cdot) \tag{B.8}$$

where

$$\tilde{\Phi}_i(\cdot) = \frac{\Phi_i(\|\mathbf{x} - \mathbf{c}_i\|_{\mathbf{\Sigma}_i})}{\sum_{j=1}^{I} \Phi_j(\|\mathbf{x} - \mathbf{c}_j\|_{\mathbf{\Sigma}_j})} \tag{B.9}$$

$$\Phi_i(r_i) = \exp(-\frac{1}{2} r_i^2) \tag{B.10}$$

So the output of the network is normalized by the sum of all (non-weighted) hidden layer neuron outputs. It has the *partition of unity* property

$$\sum_{i=1}^{I} \Phi_i(\cdot) = 1 \tag{B.11}$$

In contrast to RBF networks, typically NRBF networks are employed without offset, i.e. $w_0 = 0$, $\Phi_0(\cdot) = 0$, because the normalization allows one to fix an output level without any explicit offset value. A change in one neuron (in the center or standard deviation) affects all basis functions. Furthermore, it can be gyaranteed that the NRBF network output always lies in the interval

$$\min_i(w_i) \leq \hat{y} \leq \max_i(w_i) \tag{B.12}$$

Furthermore, the following partial derivatives are needed for the adaptation of the network

$$\frac{\partial \hat{y}}{\partial w_i} = \tilde{\Phi}_i(\cdot) \tag{B.13}$$

$$\frac{\partial \hat{y}}{\partial x_i} = \frac{\sum_{j=1}^{I} \Phi_j(\cdot) \sum_{j=1}^{I} w_j \Phi_j'(\cdot) - \sum_{j=1}^{I} w_j \Phi_j(\cdot) \sum_{j=1}^{I} \Phi_j'(\cdot)}{\left(\sum_{k=1}^{I} \Phi_k(\cdot)\right)^2} \tag{B.14}$$

$$= \frac{\sum_{j=1}^{I} w_j \Phi_j'(\cdot) - \hat{y} \sum_{j=1}^{I} \Phi_j'(\cdot)}{\sum_{k=1}^{I} \Phi_k(\cdot)} \tag{B.15}$$

$$= \frac{\sum_{j=1}^{I} (w_j - \hat{y}) \Phi_j'(\cdot)}{\sum_{k=1}^{I} \Phi_k(\cdot)} \tag{B.16}$$

$$= \frac{\sum_{j=1}^{I} (w_j - \hat{y}) \left(-\frac{x_i - c_{i,j}}{\sigma_j^2} \Phi_j(\cdot)\right)}{\sum_{k=1}^{I} \Phi_k(\cdot)} \tag{B.17}$$

APPENDIX C

RADIAL BASIS FUNCTIONS VERSUS THE MULTI-LAYER PERCEPTRON

Properties	MLP	RBF	NRBF
Interpolation behavior	+	−	+
Extrapolation behavior	0	−	+
Locality	−	++	+
Accuracy	++	0	0
Smoothness	++	0	+
Sensitivity to noise	++	+	+
Parameter optimization	−−	$++^{*}/--^{**}$	$++^{*}/--^{**}$
Structure optimization	−	+	−
Online adaptation	−−	+	+
Training speed	−−	$+^{*}/--^{**}$	$+^{*}/--^{**}$
Evaluation speed	+	0	0
Curse of dimensionality	++	−	−
Interpretation −−	0	0	
Incorporation of constraints	−−	0	0
Incorporation of prior knowledge	−−	0	0
Usage	++	0	0

$*$ = linear optimization, $**$ = nonlinear optimization,

$++/--$ = model properties are very favorable / undesirable.

Table C.1: Comparison between MLP and RBF networks [12]

BIBLIOGRAPHY

[1] L.C. Baird. Reinforcement learning in continuous time: Advantage updating. *Proceedings of the International Conference on Neural Networks*, 1994.

[2] R.J.M. Bennis. *Fuzzy Logic for the Automated Transfer Vehicle (ATV) - Adaptive Fuzzy Control by Reinforcement Learning for the Automated Transfer Vehicle (ATV).* PhD thesis, Delft Univeristy of Technology, 2000.

[3] R.J.M. Bennis. *Fuzzy Logic for the Automated Transfer Vehicle (ATV) - Architectural and Detailed Design Document.* PhD thesis, Delft Univeristy of Technology, 2000.

[4] R.J.M. Bennis. *Fuzzy Logic for the Automated Transfer Vehicle (ATV) - Simulation of Fuzzy Reinforcement Learning for the Automated Transfer Vehicle (ATV).* PhD thesis, Delft Univeristy of Technology, 2003.

[5] D. Wettergreen C. Gasskett and A. Zelinsky. Q-learning in continuous state and action spaces. *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, 1999.

[6] M.R. Coulom. *Apprentissage par renforcement utilisant des réseaux de neurones, avec des applications au contrôle moteur.* PhD thesis, l'Ecole Doctorale Ingénierie pour le Vivant : Santé, Cognition, Environnement, juin 2002.

[7] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[8] W. Fehse. *Automated Rendezvous and Docking of Spacecraft*, volume 16 of *Cambridge Aerospace Series.* Cambridge University Press, 2003.

[9] A. Ram J.C. Santamaria, R.S. Sutton. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2), 1998.

[10] T. Kohonen. The self-organising map. *Neurocomputing*, 21:1–6, 1998.

[11] R.A. Monte. The random walk for dummies.

[12] O. Nelles. *Nonlinear System Identification.* Springer, 2001.

[13] A.J. Smith. Dynamic actions in reinforcement learning. 2001.

[14] A.J. Smith. Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15:1107–1124, 2002.

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

[16] C.F. Touzet. Neural reinforcement learning for behaviour systhesis. *Robotics and Autonomous Systems*, 22:251–281, 1997.

[17] C.F. Touzet. Reinforcement function design and bias for efficient learning in mobile robots. 1998.