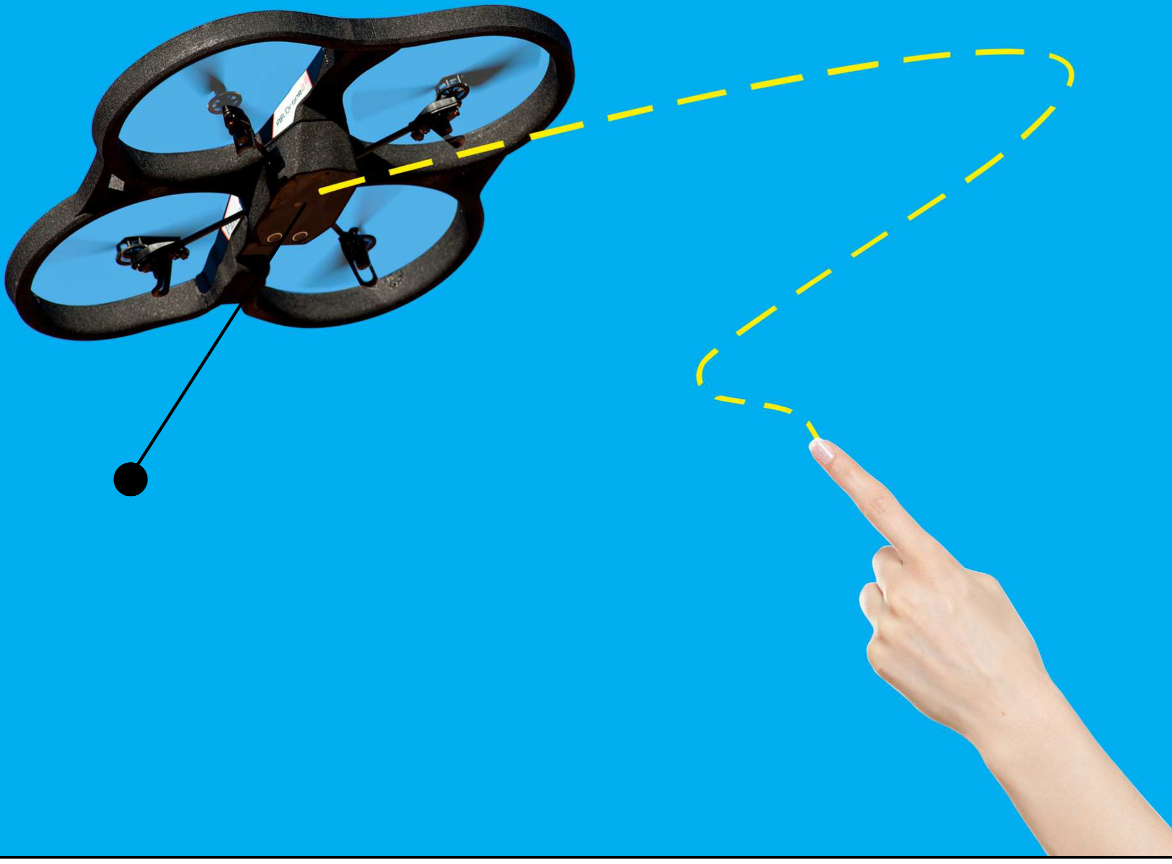
Human Demonstrations for Fast and Safe Exploration in Reinforcement Learning

G.K. Schonebaum

Master of Science Thesis



Delft University of Technology

Faculty of Aerospace Engineering

Human Demonstrations for Fast and Safe Exploration in Reinforcement Learning

Master of Science Thesis

For the degree of Master of Science in Control and Simulation at Delft University of Technology

G.K. Schonebaum

May 13, 2016

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology
Department of
Control and Operations

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled

Human Demonstrations for Fast and Safe Exploration in Reinforcement Learning

by

G.K. Schonebaum

in partial fulfillment of the requirements for the degree of

Master of Science Control and Simulation

Dated: May 13, 2016

Supervisor(s):

dr.ir. E. van Kampen

J. Junell, MSc.

Reader(s):

dr. Q.P. Chu

dr.ir. W.J.C. Verhagen

Table of Contents

Preface

Eleven months ago I walked into the office of my supervisor Erik-Jan van Kampen to discuss some potential thesis topics. He told me about a research field within machine learning, called reinforcement learning, on which he had a nice assignment available. I read some things about this topic and found claims that the way machine learning works can be sometimes surprisingly similar to the way humans learn. I started thinking about the topic, and after some time came to the conclusion that this is a field of engineering which interests me a lot. Having worked as a sailing instructor for a long time, I developed a passion for teaching. I like the challenge of explaining and teaching in such a way that my students learn as quickly as possible. Approaching this challenge from a totally different viewpoint - making a machine learn as quickly as possible - seemed like something I was going to enjoy a lot. After 10 months of diving into the world of reinforcement learning, and optimizing the teaching for the computer I found that there are actually a lot of parallels between the way we teach humans, and the methods in which we can teach machines.

Many people have consciously or unconsciously contributed to the finalization of this thesis. First of all, I would like to thank my supervisors Erik-Jan van Kampen and Jaime Junell for there guidance and help during this final phase of my studies. I have really appreciated their suggestions and the motivation I got from our meetings. I believe my level of academic English improved quite a bit, which is primarily because of the extensive reviews of Jaime. Working with an interesting reinforcement learning algorithm, I found myself helped by an expert in the field: Lucian Busoniu, who I want to thank for his detailed answers to my questions. A special mention here is also truly deserved by my good friend Tim, who was always happy to answer my questions in his field of system identification. He has a great skill of explaining difficult things in an understandable way. Furthermore, I would like to thank all the people who have reviewed my writings in all phases of this thesis project. I am certain that this help increased the clarity and readability of this thesis tremendously. Finally, I would like to thank my parents, my brothers and my girlfriend for their direct and indirect help and support during my thesis, but even more so during the rest of my studies. It have been seven great years, and I am looking forward to the next step.

Gerben Schonebaum

Delft,

May 13, 2016

“Learn from the mistakes of others. you can’t live long enough to make them all yourself”

— *Eleanor Roosevelt*

Glossary

List of Acronyms

MAV	Micro Aerial Vehicle
RL	Reinforcement Learning
HD	Human Demonstration
MDP	Markov Decision Process
DP	Dynamic Programming
SARSA	State Action Reward State Action
LA	Learning Automata
PI	Policy Iteration
AVI	Approximate Value Iteration
LSPI	Least Squares Policy Iteration
IOLSPI	Informed Online Least Squares Policy Iteration
CAFVI	Continuous Action Fitted Value Iteration
DDP	Differential Dynamic Programming
BF	Basis Function
PBF	Polynomial Basis Function
RBF	Radial Basis Function
OLS	Ordinary Least Squares
LWLR	Locally Weighted Linear Regression
PID	Proportional Integral Derivative
ISM	Integral Sliding Mode

List of Symbols

Greek Symbols

α	-	Learning rate
α	rad	Angle of rotational pendulum
χ	-	Termination criterion
δ	-	Small constant used in LSPI to ensure invertibility
ϵ	-	Exploration rate
η	rad	Slung angle
Γ	-	Matrix containing sample information (current state)
γ	-	Discount factor
Λ	-	Matrix containing sample information (next state)
μ	-	RBF center location
ω	rad/s	Quadrotor rotor speed
ϕ	-	Basis function
ϕ	rad	Roll angle
π	-	Policy
ψ	-	Chebyshev polynomial
ψ	rad	Yaw angle
$\rho(s,a)$	-	Reward function
σ	-	RBF width
σ	-	Standard deviation
τ	-	Fitting parameter for LWLR
θ	rad	Pitch angle
θ	rad	Slung angle 2D simulation
ξ	-	Variable in Chebyshev polynomial

Latin Symbols

A	-	Action space
a	-	Action
b	Nms/rad	Friction coefficient
C	-	Cut off number
c	-	Number of RBF centers
D	-	Sample set
$f(s,a)$	-	Transition function
g	m/s^2	Gravitational acceleration
J	kgm^2	Mass moment of inertia
K	-	End time of episode
K	Nm/A	Torque constant
k	-	Time step

L	m	Cable length
l	m	Pendulum length
l_s	-	Maximum number of policy iterations in offline LSPI
m	kg	Mass
n	-	Number of basis functions
n	-	Number of independent random realizations
n_s	-	Number of samples
$N_{0.01}$	-	Number of samples that are more than 1% of their original value
P	-	Transition matrix
p	m	Quadrotor position
Q	-	State-action value
R	Ω	Resistance
R	-	Return
r	-	Reward
S	-	State space
s	-	State
T_s	s	Time step 2D simulation
V	-	State value
v	m/s	Quadrotor velocity
w	-	Basis function weight
z	-	Vector containing sample information (reward)

Subscripts

g	-	Goal
k	-	Current time step
l	-	Current policy
q	-	Quadrotor

Superscripts

$*$	-	Optimal
π	-	Using policy π

Chapter 1

Introduction

In recent years the need for automation and autonomy in all sorts of vehicles and systems has been growing. Especially in the development of controllers for Micro Aerial Vehicles (MAVs) autonomy has large benefits [1]. Making such vehicles fully autonomous is difficult because it means that they have to perform their missions with no human intervention. As a result, it is necessary that the vehicle and its control system are inherently safe, and are able to adapt to changing conditions. The dynamics of some of these vehicles, in particular small and flapping wing MAVs such as the Delfly [2], are complex. For these vehicles, there is a need for model-free controllers to circumvent the problems arising in model-based controllers because of errors in modeling [3].

Reinforcement Learning (RL) is a promising method to meet these demands, since it can be applied without a model of the system dynamics. In addition, when used online, RL controllers can adapt to changing conditions [4]. In RL the controller receives scalar rewards from the environment and uses these to improve its control policy. The goal for the controller is to develop a policy that maximizes the return, which is the sum of all rewards, received from its current state onward. RL is generally referred to as a method which is applied without using prior knowledge. However, learning a policy from scratch can be a time consuming task, in which there is little control over the parts of the state space that will be visited. In many real life applications, such as flying with MAVs, this exploration could lead to unsafe situations while the controller is still learning towards a good control policy, and in the worst-case result in a crash.

The formalization of RL allows for using prior knowledge to accelerate the learning. Successful attempts have been made to include knowledge on the dynamics of the system to speed up the learning [5, 6]. These attempts used different kinds of knowledge on the system dynamics to do so. Another method to improve the speed of learning is the use of human demonstrations to explore the state space of a system. Several attempts have been made to do so [6, 7], however, these attempts did use a model of the dynamics of the system at hand. A method that accelerates the reinforcement learning in a model-free and online fashion through the use of human demonstrations, has not yet been found.

The goal of this research is to find out in which way human demonstrations can be used to reduce the learning time, and improve the safety of online, model-free reinforcement learning algorithms. In order to achieve this goal, the following research question will be answered: *How can we use demonstrations of a human to improve the performance of a reinforcement learning controller?* This question is subdivided in the following sub-questions:

- How can human demonstrations be used within RL algorithms?
- How do RL algorithms using human demonstrations compare to each other, and how do they compare to existing techniques?
- In what way should the human demonstrate in order to maximize the benefit of the demonstration for the RL controller?
- How does the effect of the human demonstration depend on the specific controlled system?

This report consists of three parts. Part I contains a conference paper that includes the used approach and the main findings and results. This first part is a stand alone document. The preliminary research carried out before the work described in the paper was done is presented in Part II. It contains a literature study in which relevant related literature is reviewed and a preliminary, two-dimensional simulation on a simple quadrotor system. Part III contains additional documentation. Chapter 8 forms the intermediate step between the 2D preliminary simulation described in Part II and the Informed Online Least Squares Policy Iteration algorithm discussed in the paper (Part I). In Chapter 9 a more detailed description of the implementation of the method in the paper is given. The complete set of results for the different simulations is presented in Chapters 10 and 11 for the rotational pendulum and the quadrotor slung load problem respectively. Finally, Chapter 12 presents the overall conclusions on the research, and reviews the research questions presented in this Introduction.

Note on the chronological structure of the report
The first part of this research is described in Part II, in which the original research questions are posed. In this early phase of the research, the goal was to implement the model-free Reinforcement Learning algorithm using human demonstration on a real quadrotor. As such the literature study is focused on RL algorithms in quadrotor applications. In addition, Part II presents a plan for 3D simulations and an experimental setup. When the implementation of the model-free RL algorithm was applied to a continuous problem however, as is described in Chapter 8, it was found that there is still so much work to do on this algorithm, that the implementation on an actual quadrotor is unfeasible in the time span of this thesis project. As such, the scope was limited to the model-free algorithm itself, and the use of human demonstrations within. In the remainder of Part III and the conference paper (Part I) this new scope is used, and as a consequence the research questions were changed to the ones presented in this introduction. As a result, the preliminary research should be read with this change of scope in mind.

Part I

Conference Paper

Human Demonstrations for Fast and Safe Exploration in Reinforcement Learning

Gerben Schonebaum[*], Jaime Junell[†] and Erik-Jan van Kampen[‡]
Delft University of Technology, Delft, The Netherlands

Reinforcement learning is a promising framework for controlling complex vehicles with a high level of autonomy, since it does not need a dynamic model of the vehicle, and it is able to adapt to changing conditions. When learning from scratch, the performance of a reinforcement learning controller may initially be poor and –for real life applications– unsafe. In this paper the effects of using human demonstrations on the performance of reinforcement learning is investigated, using a combination of offline and online least squares policy iteration. It is found that using the human as an efficient explorer improves learning time and performance for a benchmark reinforcement learning problem. The benefit of the human demonstration is larger for problems where the human can make use of its understanding of the problem to efficiently explore the state space. Applied to a simplified quadrotor slung load drop off problem, the use of human demonstrations reduces the number of crashes during learning. As such, this paper contributes to safer and faster learning for model-free, adaptive control problems.

Nomenclature

Symbols

s	State
a	Action
π	Policy
r	Immediate reward
$f(s,a)$	Transition function
$\rho(s,a)$	Reward function
R	Return
ϵ	Exploration rate
γ	Discount factor
$Q(s,a)$	State-action value
$\phi(s,a)$	Basis function
w	Basis function weight
η	Forget factor
K_θ	Update interval
Γ, Λ, z	Matrices containing sample information

Abbreviations

MAV	Micro aerial vehicle
RL	Reinforcement learning
LSPI	Least squares policy iteration
IOLSPI	Informed online least squares policy iteration
HD	Human demonstration
BF	Basis function
RBF	Radial basis function

I. Introduction

In recent years the need for automation and autonomy in all sorts of vehicles and systems has been growing. Especially in the development of controllers for Micro Aerial Vehicles (MAVs) autonomy has large benefits.[1] Making such vehicles fully autonomous is difficult because it means that they have to perform their missions with no human intervention. As a result, it is necessary that the vehicle and its control system are inherently safe, and are able to adapt to changing conditions. The dynamics of some of these vehicles, in particular small and flapping wing MAVs such as the Delfly,[2] are complex. For these vehicles, there is a need for model-free controllers to circumvent the problems arising in model-based controllers because of errors in modeling.[3]

[*]Master Student, TU Delft Aerospace Engineering, Control & Simulation. Delft. The Netherlands.
[†]PhD Candidate, TU Delft Aerospace Engineering, Control & Simulation. Delft. The Netherlands, Student Member AIAA.
[‡]Assistant Professor, TU Delft Aerospace Engineering, Control & Simulation. Delft. The Netherlands, Member AIAA.

Reinforcement Learning (RL) is a promising method to meet these demands, since it can be applied without a model of the system dynamics. In addition, when used online, RL controllers can adapt to changing conditions.[4] In RL the controller receives scalar rewards from the environment and uses these to improve its control policy. The goal for the controller is to develop a policy that maximizes the return, which is the sum of all rewards, received from its current state onward. RL is generally referred to as a method which is applied without using prior knowledge. However, learning a policy from scratch can be a time consuming task, in which there is little control over the parts of the state space that will be visited. In many real life applications, such as flying with MAVs, this exploration could lead to unsafe situations while the controller is still learning towards a good control policy, and in the worst-case result in a crash.

The formalization of RL allows for using prior knowledge to accelerate the learning. Successful attempts have been made to include knowledge on the dynamics of the system to speed up the learning.[5,6] These attempts used different kinds of knowledge on the system dynamics to do so. Another method to improve the speed of learning is the use of human demonstrations to explore the state space of a system. Several attempts have been made to do so,[6,7] however, these attempts did use a model of the dynamics of the system at hand. A method that accelerates the reinforcement learning in a model-free and online fashion through the use of human demonstrations, has not yet been found.

The goal of this research is to find out in which way human demonstrations can be used to reduce the learning time, and improve the safety of online, model-free reinforcement learning algorithms. The RL algorithm proposed in this research is purely model-free, which is useful for vehicles with complex dynamics such as MAVs. In addition, it uses basis functions to approximate the state-action values which allows for the application to continuous state systems.[8] The algorithm uses a combination of two state of the art RL techniques: Offline Least Squares Policy Iteration[9] (LSPI) that is able to use the information of the human demonstration sample, and online LSPI[10] which continues learning online, and hence allows to adapt to changing conditions. This combined algorithm is called Informed Online Least Squares Policy Iteration (IOLSPI).

The contribution of this research is the implementation of this model-free and online IOLSPI algorithm, resulting in faster and safer exploration in RL. To test the proposed algorithm, a standard RL benchmark task is selected: the rotational pendulum. In this standard problem it is shown that the learning time could be significantly reduced by using human demonstrations in IOLSPI. Furthermore, it is shown that while using human demonstrations, the algorithm is still able to adapt to changing conditions. However, it is found that the benefit of the human demonstrations is not the same for each problem.

The rotational pendulum problem includes no intuitive aspect of safety (there is no clear unsafe part of the state space), therefore, an additional problem is reviewed: the control of a quadrotor which carries a slung load. This combined vehicle has complex dynamics. In a task of dropping a load in a target, a capable human expert could demonstrate good behavior. In this example the relevance of safety in exploring becomes more apparent, since faulty policies result in dropping the load at the wrong location, or crashing the quadrotor. It is shown that using human demonstrations the number of crashes while learning is reduced for this quadrotor slung load problem. Reinforcement learning has already been applied to quadrotors with a slung load.[3,11–13] All these studies use a model of the system dynamics. In this paper, a two-dimensional version of this slung load problem is used with a model-free reinforcement learning controller.

This paper is structured as follows: In section II, a short background on reinforcement learning and some of its basic concepts and notations are introduced. Section III describes the used reinforcement learning algorithm and methodology. Section IV gives an overview of the most important results, and section V presents the conclusions of the paper.

II. Reinforcement Learning Preliminaries

In reinforcement learning a Markov Decision Process is considered, with a state space S and an action space A. The RL agent starts in state s_k and takes action a_k according to its current policy π_l:

$$a_k = \pi_l(s_k) \tag{1}$$

The transition function f, which describes the dynamics of the system, gives the next state s_{k+1}:

$$s_{k+1} = f(s_k, a_k) \tag{2}$$

Furthermore, the agent receives a reward r_{k+1} which follows from the reward function ρ:

$$r_{k+1} = \rho(s_k, a_k) \tag{3}$$

The goal for the agent is to find a policy π that maximizes the expected return R in each state. In Eq. (4) the return is defined.

$$R^{\pi_l}(s_0) = \sum_{k=0}^{K} \gamma^k \rho(s_k, \pi_l(s_k)) \tag{4}$$

Where K is the number of time steps and $\gamma \in [0, 1]$ is the discount factor, ensuring that the further in the future a reward is received, the less it contributes to the return at the current state. The expected return for a certain policy π_l, starting from state s_k, applying action a and using policy π_l onward is given by the Q-function:

$$Q^{\pi}(s_k, a) = E\{R^{\pi}(s_k)|a\} \tag{5}$$

Using the Q-function, the greedy policy for that Q-function can be found with the following equation:

$$\pi_{l+1}(s_k) = \arg\max_a Q^{\pi_l}(s_k, a) \tag{6}$$

In many practical applications, the state and action spaces are large or uncountable, and as such it becomes intractable to describe the Q-value for each combination of state and action separately.[8] In these cases an approximation of the Q-value is made using a function approximator. Usually a linear combination of basis functions (BFs) $\phi(s, a)$ is used. These BFs are weighted with a tuning vector w_l to approximate the Q-values $\hat{Q}(s, a)$:

$$\hat{Q}(s, a) = \phi^T(s, a) w_l \tag{7}$$

In policy iteration algorithms two steps are alternated. First the policy is evaluated, i.e. the Q-function is determined. This is done by solving the Bellman equation (Eq. (8)).[14] After this evaluation step the policy is updated using equation (6). This new policy is then evaluated again, and so on.

$$Q^{\pi_l}(s_k, a) = \rho(s_k, a) + \gamma Q^{\pi_l}(f(s_k, a), \pi_l(f(s_k, a))) \tag{8}$$

In a model-free algorithm, the Bellman equation cannot be solved since the reward function $\rho(s_k, a_k)$ and transition function $f(s_k, a_k)$ are unknown. However, the transition and reward information contained in a set of samples D collected for the system can be used to approximate the Bellman equation. D contains n_s samples constituted of $\{s_k, a_k, r_{k+1}, s_{k+1}\}$. The Bellman equation can be formulated in matrix sense for a set of samples D (Eq. (9)). For a more elaborate discussion on this approach we refer to the paper by Lagoudakis.[9]

$$\Gamma w_l = z + \gamma \Lambda w_l \tag{9}$$

The matrices Γ and Λ and vector z in Eq. (9) are defined by Eqs. (10) - (12), and contain the information in the transition samples.

$$\Gamma_{k+1} \leftarrow \Gamma_k + \phi(s_k, a_k)\phi^T(s_k, a_k) \tag{10}$$

$$\Lambda_{k+1} \leftarrow \Lambda_k + \phi(s_k, a_k)\phi^T(s_{k+1}, \pi_l(s_{k+1})) \tag{11}$$

$$z_{k+1} \leftarrow z_k + \phi(s_k, a_k) r_{k+1} \tag{12}$$

To approximate the Q-function for a certain policy, the least square error in the modified Bellman equation (Eq. (9)) is minimized.[14] A policy evaluation algorithm called LSTD-Q[9] (Least Squares Temporal Difference for Q-functions) solves Eq. (9), which can be slightly modified to Eq. (13):

$$\frac{1}{n_s}\Gamma w_l = \frac{1}{n_s} z + \frac{1}{n_s}\gamma \Lambda w_l \tag{13}$$

This modified equation is mathematically not different from Eq. (9), but improves the numerical stability.[8]

After the policy evaluation, the policy is updated using Eq. (6). The new policy is evaluated again using LSTD-Q. This policy iteration method is called Least Squares Policy Iteration.[9]

The approach described has an interesting characteristic. The policy is defined implicitly by the Q-function: for each state the greedy action maximizes the Q-value (Eq. (6)). In practice the greedy action only needs be calculated for the state in which the action has to be taken, and not for the whole state space. This approach makes a method to explicitly define the policy for each state redundant. Since the policy is defined with respect to the Q-value (Eq. (6)) and an approximation of the Q-value with BFs is used, the policy is at any moment fully defined by the current BF weights w_l.

III. Least Squares Policy Iteration

In this paper the Least Squares Policy Iteration (LSPI) algorithm is used. There are different ways described in literature in which this algorithm has been implemented. Lagoudakis and Parr[9] invented the algorithm for offline use. Busoniu et al. altered the algorithm to use it online. This online algorithm will be referred to as Online Least Squares Policy Iteration[10] (OLSPI). Both these approaches will form the foundation for the integrated algorithm that is proposed in this paper, which uses offline LSPI to learn from a batch of pre-collected samples, and afterwards continues with gathering samples online. This combined algorithm is called Informed Online Least Squares Policy Iteration (IOLSPI).

A. Offline LSPI algorithm

Algorithm 1 shows the offline LSPI algorithm.[9] This algorithm uses a batch of samples to learn a new policy. These samples can be generated using a simulator, or they can be obtained as recordings during the use of the physical system at hand. This algorithm starts with no knowledge of the problem or the policy in advance. The matrix Λ_0 and vector z_0 are thus initialized with zeros. The matrix Γ_0 is the identity matrix multiplied with a small constant δ, to make sure it is invertible. In line 3, the policy weights are randomly initialized. This random initialization gives the need to run the algorithm for several random seeds and average its results. In lines 6-8, the complete batch of samples is processed according to equations (10) - (12). After the whole batch is processed, the policy update follows in line 10. This process is repeated until either the policy has converged, such that the difference in weights has dropped below the stopping criterion χ, or a maximum number of iterations l_{max} is reached. The output of this algorithm is a policy which uses the information of the presented sample batch. Note that the number of samples is directly linked to the computation time needed to run the algorithm.

Algorithm 1 Offline LSPI[9]

```
1:  **Input:** $n$ BFs $\phi_1(s,a),\ldots,\phi_n(s,a)$; $\gamma$; $\delta$; Sample set $D$ with $n_s$ samples; $\chi$
2:  $\Gamma_0 \leftarrow \delta I_{nxn}$ ; $\Lambda_0 \leftarrow 0_{nxn}$; $z_o \leftarrow 0_{nx1}$
3:  $l \leftarrow 0$; initialize weights (policy) $w_0$
4:  **while** $l < l_{max}$ and $\Delta w > \chi$ **do**
5:      **for** $k = 0 : n_s$ **do**
6:          $\Gamma_{k+1} \leftarrow \Gamma_k + \phi(s_k,a_k)\phi^T(s_k,a_k)$
7:          $\Lambda_{k+1} \leftarrow \Lambda_k + \phi(s_k,a_k)\phi^T(s_{k+1},\pi_l(s_{k+1}))$
8:          $z_{k+1} \leftarrow z_k + \phi(s_k,a_k)r_{k+1}$
9:      **end for**
10:     $w_{l+1} \leftarrow$ solve $\frac{1}{k+1}\Gamma_{k+1}w_l = \gamma\frac{1}{k+1}\Lambda_{k+1}w_l + \frac{1}{k+1}z_{k+1}$
11:     $\Delta w = ||w_l - w_{l+1}||$
12:     $l \leftarrow l+1$
13: **end while**
```

B. Online LSPI algorithm

Busoniu et al. have extended the offline LSPI algorithm to a variant that can be used online.[10] In online LSPI the algorithm gathers its samples online to update the policy, contrary to offline LSPI. In Algorithm 2 the working principle of Online LSPI is shown. It is similar to Algorithm 1, but some clear differences are present. Since the algorithm runs online, it starts to update its policy after having processed a couple of samples, contrary to offline LSPI, which updates its policy only after having processed all of its samples. In online LSPI this policy update happens every K_θ time steps (line 10). The larger this update interval K_θ is, the longer the algorithm waits before the policy is updated. On the other hand, when K_θ becomes small, ultimately 1, this means that after each time step the policy is updated, which makes the algorithm slower. With a K_θ of 1 the update is called fully optimistic.[15] In general a partially optimistic update is used.[11]

Another feature of the online LSPI algorithm is that there is the need for exploration, as for all online RL algorithms.[16] Without exploration, the agent would always take the same action at the same state, and as such the estimate of the Q-function would be poor. Furthermore, exploration allows gathering samples in all parts of the state space, which improves the Q-function estimate. As can be seen in line 5 of algorithm 2, at each time step, with a probability of ϵ a random (exploratory) action is taken. Such a policy is called an ϵ-greedy policy.[4] If $\epsilon = 1$, the action is chosen fully at random, if $\epsilon = 0$, the action is chosen fully greedy. Different schemes for this exploration rate can be used. In online LSPI it makes sense to start with a high exploration rate, and make it decay over time. In this way at the beginning, when the policy is not good yet, many different samples are gathered to improve the Q-function estimate. Gradually it becomes more exploiting, to get better performance. The exploration rate should not decay to zero (fully greedy) however, since then the algorithm is not able to adapt to changes in the conditions. An example of an exponential decaying exploration rate that could be used is presented in Eq. (14).[10]

$$\epsilon_k = \epsilon_0 \epsilon_d^{kT_s} \tag{14}$$

In this equation ϵ_0 is the initial exploration rate, $\epsilon_d \in [0, 1]$ the exponential decay factor and T_s the sampling time.

Following from the need for adaptation to changing conditions, in addition to the algorithm presented in [10], a forget factor η is added (line 7-9). This forget factor makes sure that recently gathered samples have a higher impact on the policy than old ones. This factor is added to make sure that the algorithm is able to change along with changes in conditions by not carrying along too much outdated information.

Algorithm 2 Online LSPI[10,11] with ϵ-greedy exploration and forget factor

```
1:  **Input:** $n$ BFs $\phi_1(s,a), \dots, \phi_n(s,a)$; $\gamma$; $\{K_\theta\}_{k\geq 0}$; $\{\epsilon_k\}_{k\geq 0}$; $\Gamma_0$; $\Lambda_0$; $z_0$; $w_0$
2:  $l \leftarrow 0$
3:  Measure initial state $s_0$
4:  **for** every time step $k = 0, 1, 2, \dots$ **do**
5:      $a_k \leftarrow \begin{cases} \text{exploit: } \arg\max_a \phi^T(s,a) w_l \text{ with prob } 1-\epsilon_k \\ \text{explore: random action with prob } \epsilon_k \end{cases}$
6:      Apply $a_k$ and observe next state $s_{k+1}$ and reward $r_{k+1}$
7:      $\Gamma_{k+1} \leftarrow \eta\Gamma_k + \phi(s_k, a_k)\phi^T(s_k, a_k)$
8:      $\Lambda_{k+1} \leftarrow \eta\Lambda_k + \phi(s_k, a_k)\phi^T(s_{k+1}, \pi_l(s_{k+1}))$
9:      $z_{k+1} \leftarrow \eta z_k + \phi(s_k, a_k) r_{k+1}$
10:     **if** $k = (l+1)K_\theta$ **then**
11:         $w_l \leftarrow$ solve $\frac{1}{k+1}\Gamma_{k+1} w_l = \gamma \frac{1}{k+1}\Lambda_{k+1} w_l + \frac{1}{k+1} z_{k+1}$
12:         $l \leftarrow l + 1$
13:     **end if**
14: **end for**
```

An advantage of the online LSPI algorithm is that the computational effort to update the policy is not dependent on the number of samples processed. As such the time the online LSPI algorithm has been running has no influence on the required computation time for the policy updates.

It should be noted that this algorithm works under the assumption that the Q-function does not change too much for subsequent policies.[10] This means that the previous values of the matrices Λ and Γ and the z vector are also representative for the next policy. Using the forget factor does change this assumption a little, since the part in these matrices corresponding to the samples that already have been forgotten, does not influence the Q-function anymore.

C. Informed Online LSPI algorithm

In this paper the online and offline LSPI algorithms are combined. Using a set of samples, which can either be generated with a simulator, or obtained from a human demonstration on the physical system, the offline algorithm is run to find a satisfactory policy. Additionally, the samples processed offline are contained in the Λ and Γ matrix and the z vector. After running the offline LSPI algorithm, the online LSPI algorithm is used. In this Informed Online Least Squares Policy Iteration (IOLSPI) algorithm, contrary to regular online LSPI, the policy found using offline LSPI is used as initial policy for the online part of the algorithm. Furthermore, the Λ and Γ matrix and the z vector that were found after having run the offline LSPI are used instead of an empty initialization as is done in regular online LSPI. The information of the offline samples is contained in the matrices, and as such it does not start to learn from scratch. Moreover, the online algorithm still is able to learn from the samples gathered online, and as such is able to adapt to changing conditions.

IV. Experimental Study

The IOLSPI algorithm is tested on two experimental studies. The first problem setup studied is a rotational pendulum, with two state and one action variable. This is a common benchmark problem for reinforcement learning controllers. The second application is a simplified representation of a quadrotor with a slung load, which has four state and one action variable. This application is used to show the safety aspect in learning and to evaluate the performance of the algorithm for high-dimensional problems. The performance for the example problems is evaluated in two ways. Firstly, the intermediate policies are evaluated using a simulator, while learning and exploration are turned off. The discounted return is calculated, for each of the initial conditions. The discounted return for all different initial conditions is averaged. Secondly, the sum of the rewards gathered per episode while learning is presented to show the performance of the controller online. The results show the quality of the found policies. In the quadrotor problem, a third performance measure is added: The number of crashes while learning, which quantifies the safety.

A. Rotational Pendulum

The rotational pendulum is the same system as described by Busoniu et al.,[8] see figure 1. It consists of a DC motor with a horizontal spinning axis, with an eccentric mass attached. The goal is to make the pendulum swing up the mass and stabilize it pointing upwards. The state contains the angle and the angular rate: $s = [\alpha, \dot{\alpha}]^T$. The angle α is zero radians when the mass is pointing up, and π rad when it is hanging down in the middle. At the exact bottom, α transitions from π rad to $-\pi$ rad. The maximum angular rate of the rotational pendulum is 15 π rad/s.

There are three possible actions for this system: Apply a clockwise torque (+3V), a counterclockwise torque (-3V), or no torque (0V). Not for all states the torque of the motor is sufficient to push the pendulum upright in one go. For some states, for example hanging down, back and forth swings are needed to swing up the mass to the upright position. The rotational pendulum has the following properties: The mass moment of inertia of the disc J is 1.91 e-4 kgm^2, the mass of the pendulum m is 0.055 kg. The distance between the shaft and the mass l is 0.042 m, friction coefficient b = 3 e-6 Nms/rad, torque constant $K = 0.0536$ Nm/A and the internal resistance of the motor $R = 9.5\ \Omega$. The gravitational acceleration g is assumed to be 9.81 m/s^2. The equation of motion of the rotational pendulum is given by Eq. (15).

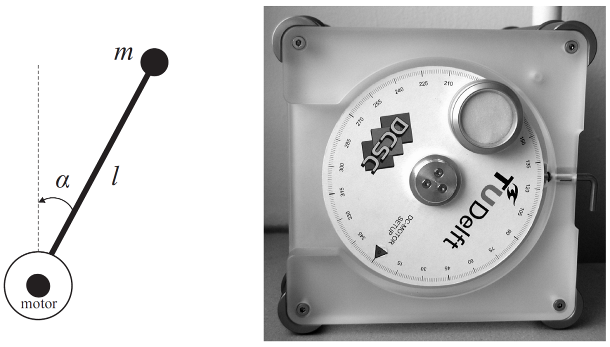


Figure 1: Schematic rotational pendulum (left), real rotational pendulum (right)[10]

$$\ddot{\alpha} = \frac{m \cdot g \cdot l \sin\alpha - b\dot{\alpha} - \frac{K^2\dot{\alpha}}{R} + \frac{K \cdot a}{R}}{J} \tag{15}$$

This problem will be simulated for 600 s, with episodes of 1.5 s, after which the state is reinitialized.

The sampling time is 0.005 s. The initial conditions (ICs) α_0 and $\dot{\alpha}_0$ used during learning are randomly selected from the following sets: $\alpha_0 = \{\pi - 0.1,\ \pi,\ \pi + 0.1\}$ rad, and $\dot{\alpha}_0 = \{-\pi,\ \frac{-2\pi}{3},\ \frac{-\pi}{3},\ 0,\ \frac{\pi}{3},\ \frac{2\pi}{3},\ \pi\}$ rad/s. With this set of initial conditions the pendulum always starts around the hanging down position with a low angular velocity, and as such always needs multiple swings to reach the goal state. Because of the partly random character of the algorithm, each result in this section is repeated 50 times with different random initializations.

In this problem a reward structure is used that penalizes an offset of the angle with respect to the goal, the angular rate and the control action (Eq. (16)). The values in this reward function were found trying different combinations, and shows a behavior that is mostly inclined to limit the error in angle.

$$r_{k+1} = s^T \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix} s + 0.01a^2 \tag{16}$$

To approximate the Q-function, an equidistant grid of 11x11 radial basis functions (RBFs) is used. This means that the vector of RBFs is $\bar{\phi}(s) = [\phi_1(s), \ldots, \phi_{121}(s)]^T$. To make state-action BFs, the RBFs are repeated for the three different actions. The RBFs are evaluated for the current action and zero for the RBFs associated with the other actions. As such the vector of 121x3 state-action BFs can be represented as follows: $\phi(s,a) = [\mathcal{I}(a=-3) \cdot \phi^T(s), \mathcal{I}(a=0) \cdot \phi^T(s), \mathcal{I}(a=3) \cdot \phi^T(s)]^T$. Here the indicator function $\mathcal{I}$ is 1 when its argument is true, and zero otherwise.

1. *Effect of tuning parameters for the online LSPI algorithm*

First the online LSPI algorithm is studied with no use of human demonstration or other prior information to get a baseline comparison. Figures 2 and 3 show the influence of different constant exploration rates. The exploration rates are kept constant for clear comparison with the Informed Online LSPI results presented later on. In figure 2 the discounted return when evaluating the intermediate policies is presented. In this figure it can be seen that to reach the optimal policy, at least some exploration is needed. Furthermore, it shows that greediness also pays off, since for this problem it results in becoming able to swing up the pendulum. As a result, samples are also gathered in the swung up part of the state space, which is beneficial for the Q-function approximation. In figure 3 the learning score is displayed, which is the summed reward per episode. Each episode is 1.5 s. In this figure it can be seen that a high exploration rate yields low learning scores, which is caused by the high percentage of random actions. Some exploration is beneficial though. As was seen in figure 2, this pays off for finding a good policy.

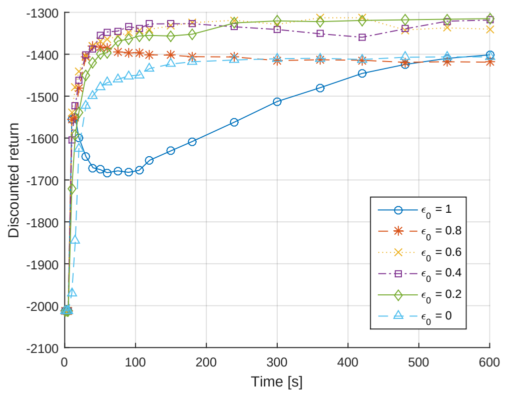


Figure 2: Online LSPI: Discounted return as a function of time for different, constant exploration rates, $K_\theta = 1000$

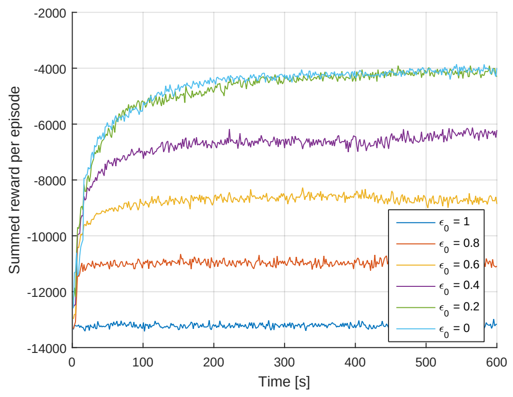


Figure 3: Online LSPI: Learning score as a function of time for different, constant exploration rates, $K_\theta = 1000$

The behavior of the policies for $\epsilon = 1$ follows a different trend. After gathering more samples, the performance first decreases, after which it increases. An explanation for the behavior is as follows. The first samples gathered contribute to finding a better policy. However, a big clutter of samples will emerge around the hanging down position, since it is hard for a purely random controller to reach the region of the state space where the pendulum is pointing up. As a result, the fitting of the Q-function will be poor because of the minimal variation in data points. Over time the chance of ending up in the upright part of the state space grows. As such the policy gradually goes up. Moreover, full randomness is not a real interesting case to consider, since in practice it would not be used anyway as can be seen in figure 3: it performs poor online.

The effect of the other tuning parameters: update interval K_θ , and discount factor γ are included in the appendix.

2. Description of human demonstrations

For the Informed Online LSPI, five different sets of human demonstrations (HDs) are used to see whether human demonstrations could improve the performance, and which type of human demonstration works best. The human demonstration samples are gathered in 21 episodes of 1.5 s, starting at each of the possible initial conditions from the previously defined grid. With a sampling time of 0.005 s, this results in sets of 6300 samples each. The human demonstrator has practiced with controlling the pendulum, and as such can be considered an "expert". Table 1 shows the details on the different sets. In some of the sets some corruption on the action is added. This means that during the human demonstration, a percentage of the actions of the human is replaced by a random action ($a \in \{-3, 0, 3\}$). This corruption can be seen as a forced exploration within the human demonstration set. A human tends to make long streaks of the same action. To swing the pendulum up for example, a human would create a large streak of samples with a clockwise action, after which a large streak of counterclockwise actions is executed, and so on. This results in a low variety of samples, since most adjacent samples contain the same action. As such, this corruption results in an increase in variety in the sample set. Furthermore, two different modes of human demonstration are used: Trying to perform as good as possible, and trying to efficiently explore the state space. The case of having 50% corruption while trying to explore was not included, since the high corruption makes it impossible to efficiently explore the whole state space.

Table 1: Overview of human demonstration sample sets, used for the results in figures 4 and 5.

Sample set number	Corruption	Description
1 (figure 6)	0%	Perform
2	0%	Explore
3	50%	Perform
4	33%	Perform
5 (figure 7)	33%	Explore

Figures 4 and 5 show the results for the different human demonstration sets used in the Informed Online Least Squares Policy Iteration (IOLSPI) algorithm. It can be seen that human demonstration set number 5 (33% percent corrupted, exploratory) performs best. Set number 3 (50% corrupted) also performs well. Both of them converge to a good policy, and show that they perform well from the start, in contrast to the non-initialized online LSPI algorithm that needs some time to learn a good policy. The uncorrupted sets perform significantly worse.

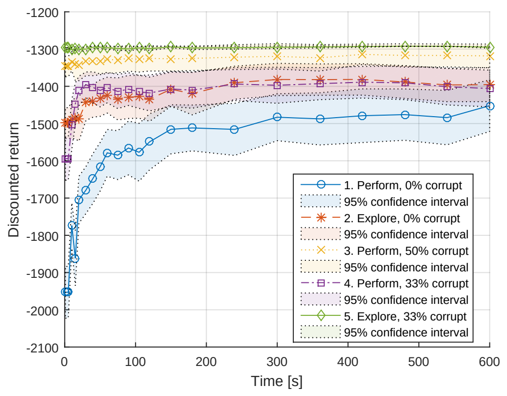


Figure 4: IOLSPI: Discounted return as a function of time for different human demonstration sets, $K_\theta = 1000$, $\epsilon = 0.2$

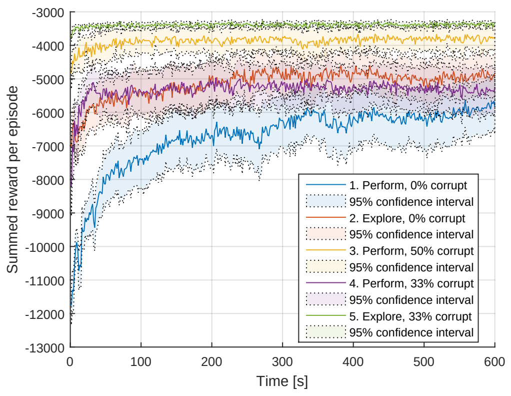


Figure 5: IOLSPI: Summed reward per episode as a function of time for different human demonstration sets, $K_\theta = 1000$, $\epsilon = 0.2$

Figures 6 and 7 show the distribution of samples for human demonstration set 1 and 5. Both the corruption in the actions and the exploration can clearly be distinguished in the latter, whereas the former shows a limited distribution of samples, both in action and in state. It can be concluded that the spread in the sample set is an important factor for the performance of a human demonstration.

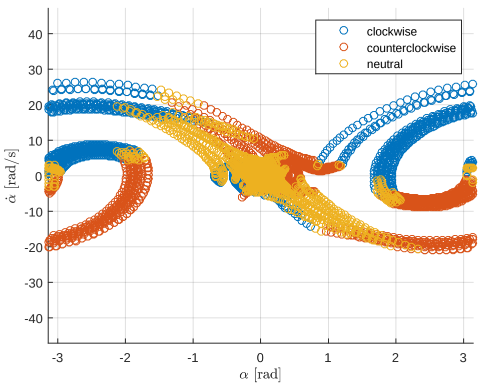


Figure 6: Sample distribution for human demonstration sample set number 1 (0% corruption, perform, 6300 samples)

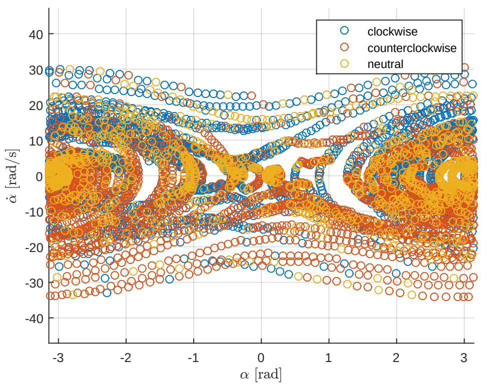


Figure 7: Sample distribution for human demonstration sample set number 5 (33% corruption, explore, 6300 samples)

To study the benefit of using human demonstrations, the best human demonstration set (number 5) is compared to non-initialized online LSPI, and to two other types of sample sets containing the same number of samples. The first is a set of non-sequential random generated samples, which requires a simulator. The sample distribution is depicted in figure 9. The other sample set is obtained using a purely random policy throughout complete episodes, resulting in sequential samples. This approach does not require a simulator, since this can be performed on the physical system. The spread in samples for this random sequential set is limited, as can be seen in figure 8. The difference of this set with the human demonstration sample sets (figures 6 and 7) and the non-sequential random sample set (figure 9) is clear. This is because using purely random actions, the chance is small to reach the swung up region of the state space, while humans use their prior intuition to reach that region.

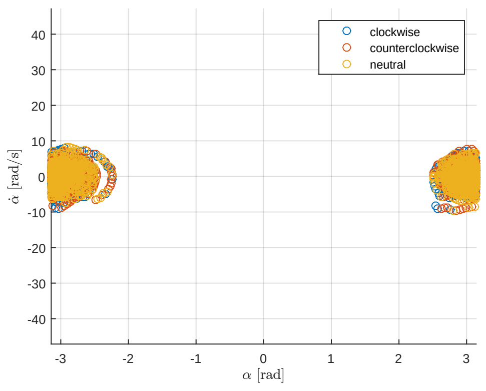


Figure 8: Sample distribution for the sample set generated using a random policy and thus sequential states (6300 samples)

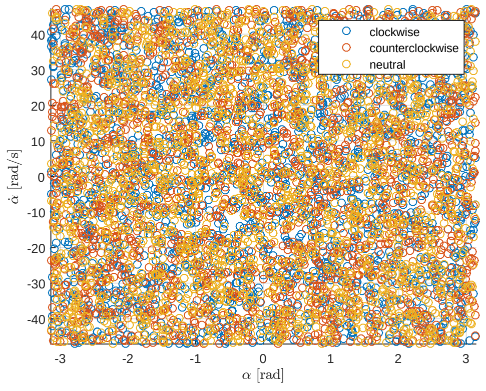


Figure 9: Sample distribution for random generated non-sequential samples (6300 samples)

Each sample set is tested with the IOLSPI algorithm. Their performance is compared to the non-initialized online LSPI algorithm. In the results of figures 10 and 11 it can be seen that the non-sequential random sample set, together with the human demonstration sample set perform best (note that the non-sequential random sample set requires a simulator). A good policy is followed from the start on, and the converged policy is also better than the converged policy found with the sequential random sample set and the non-initialized online LSPI algorithm. This graph hence shows the benefit of the use of the human demonstration for this rotational pendulum problem. Comparing the human demonstration sample set (figure 7) to the set of samples using a purely random sequence of actions in each episode (figure 8) shows why the human scores better than a random policy. The human is able to use its intuitive understanding of the problem to efficiently reach a large portion of the state space, which improves the Q-function approximation and hence the policy.

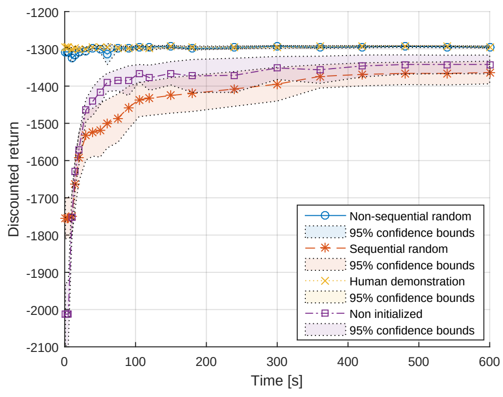


Figure 10: (Informed) Online LSPI: Discounted return as a function of time for different types of sample sets, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

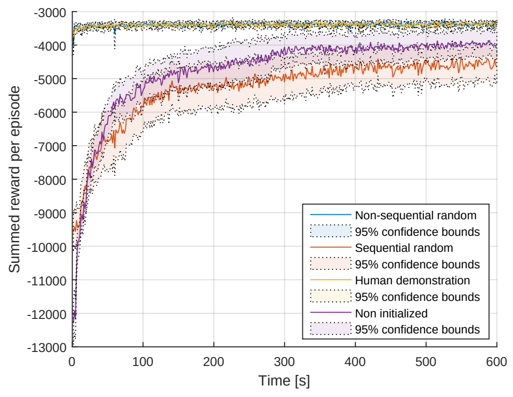


Figure 11: (Informed) Online LSPI: Learning score as a function of time for different types of sample sets, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

3. *Results for the rotational pendulum problem with distributed initial conditions*

In the previous section it was shown that for the rotational pendulum problem, when an episode always starts at a hanging down position, the human demonstration has a great benefit since the human can use its intuitive understanding of the problem to efficiently explore the state space. In problems where the entire state space is easily visited anyway, the benefit of the human demonstrations might be less pronounced. This hypothesis is tested using the same pendulum setup, but with different initial conditions. Instead of always starting from a hanging down position, the pendulum starts at different positions throughout the whole state space.

The new initial conditions (ICs) used during learning are a random combination of angle and angular velocity, selected from the following subsets: $\alpha_0 = \{-\pi,\ -\pi/2,\ 0,\ \pi/2\}$ rad, and $\dot{\alpha}_0 = \{-10\pi,\ -3\pi,\ -\pi,\ 0,\ \pi,\ 3\pi,\ 10\pi\}$ rad/s.

For this modified, "distributed ICs" problem, regardless of the policy followed, samples will be gathered throughout the entire state space. To study the impact of using human demonstrations on the IOLSPI algorithm for this modified problem, the same analysis is done as was shown in figures 10 and 11. The sequential random sample set, the non-sequential random sample set and human demonstration set number 5 are used in the IOLSPI algorithm and compared to the non-initialized online LSPI algorithm. The results in figures 12 and 13 show the difference in trend of this modified problem with the original "hanging down ICs" problem which was depicted in figures 10 and 11. It can be seen that the benefit of the human demonstration is a lot less. For the random policy and the non-initialized online LSPI the performance reaches the performance of the human demonstration sample set within 100 s. The final policy performs a little less, the difference is small however. Studying the learning score, it can be seen that within 100 s the score is for all different sample sets similar. It can be concluded that, for a problem that does not need intelligent exploration, the benefit of the human demonstration is small.

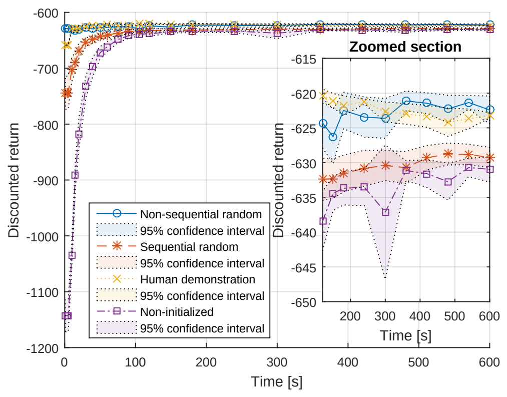


Figure 12: (Informed) Online LSPI: Discounted return as a function of time for different types of sample sets, $K_\theta = 1000$, $\epsilon = 0.2$, "distributed ICs" (Note the zoomed section from 120 s to 600 s, included to emphasize the different converged policies)

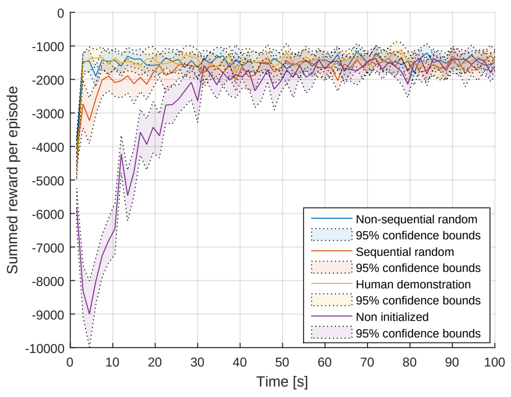


Figure 13: (Informed) Online LSPI: Learning score as a function of time for different types of sample sets, $K_\theta = 1000$, $\epsilon = 0.2$, "distributed ICs" (Only first 100 s are plotted, since after this no significant changes occur)

4. *Algorithm behavior in changing conditions*

One of the benefits of using RL for control is that it is able to cope with changing conditions. With the use of pre-generated samples within the LSPI however, there is a risk that when the conditions change, the algorithm will use too much information of previous conditions. Samples gathered in outdated conditions are no longer valid, and might prevent the algorithm from adequately keep up with the changing conditions. To check this, a sudden change in conditions has been applied to the rotational pendulum problem, both for online LSPI and for IOLSPI using a human demonstration sample set. To simulate a change in conditions, the gravitational acceleration was suddenly changed after 300 s from 9.81 m/s^2 to 19.81 m/s^2.

Figures 14 and 15 show the performance with this change in conditions. It can be seen that after the change in conditions, the score drops drastically. For both the online LSPI and the IOLSPI using the human demonstration samples, the scores go up after the change again, so it is clear that the agent is adapting to the change in conditions and develops a policy suited for the new situation.

For the IOLSPI it can be seen that after the change some jumps in the summed reward per episode occur. The policy fluctuates severely between 300 s and 340 s, which results in the summed reward per episode going up and down up to 1000 units, which is reflected in figure 15. In this region, every now and then the pendulum reaches the upright regime, at which it already has a fine policy. For the upright regime, the influence of the gravitational acceleration is lower than in the hanging down regime, since it does not includes swinging up. As such, the required change in policy when g changes is low in this regime. After gathering more and more samples over time in the new conditions, the samples gathered in the new conditions become more dominant, reshaping the policy over the whole state space. As a result, the policy is also good at the swing up regime, allowing the pendulum to always swing up. This fluctuating behavior is not seen for the online LSPI, since that policy is not yet good enough to swing up to the upright regime anyway.

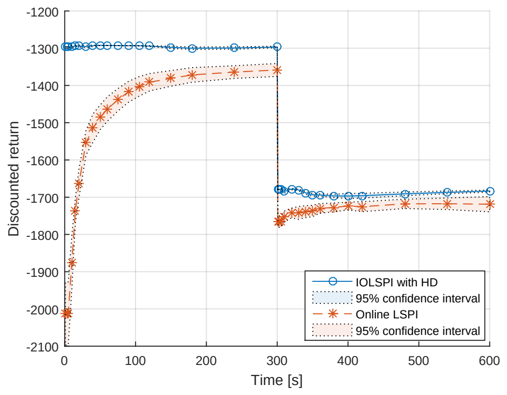


Figure 14: Discounted return as a function of time with a jump in g from 9.81 to 19.81 m/s^2 at 300 s, $K_\theta = 1000$, $\epsilon = 0.05$, using "hanging down ICs"

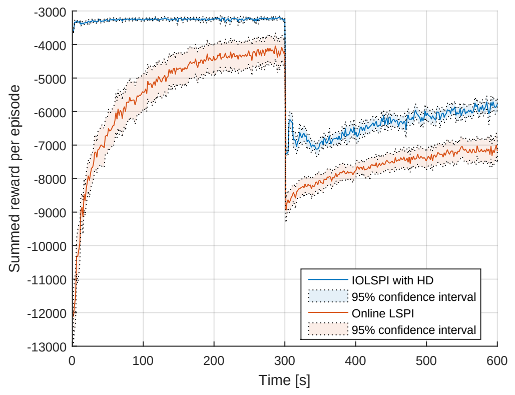


Figure 15: Learning score as a function of time with a jump in g from 9.81 to 19.81 m/s^2 at 300 s, $K_\theta = 1000$, $\epsilon = 0.05$, using "hanging down ICs"

In figures 16 and 17 the time history of the angle α and the reward is shown, corresponding to the results in figures 14 and 15. It can be seen that after the conditions change at 300 s, at first the controller performs bad, and does not succeed in swinging up the pendulum. Looking at the final two episodes however (from 597 s to 600 s), it can be seen that a satisfactory policy was learned. This clearly shows the learning effect, even after changing the conditions.

Comparing figures 16 and 17, the difference between the online LSPI with and without human demonstrations can be seen. First of all it can be seen that before the conditions change, the IOLSPI finds a good policy. The online LSPI on the other hand succeeds in only one of the two displayed episodes, and hence is not yet at an optimal policy. This conclusion is supported by figure 15 which shows that the performance of online LSPI is lower than IOLSPI. After the conditions change, at the last two episodes (from 597 s to 600 s), it is seen that both algorithms succeed in swinging up the pendulum. However, the IOLSPI needs one swing less, and reaches the goal state faster.

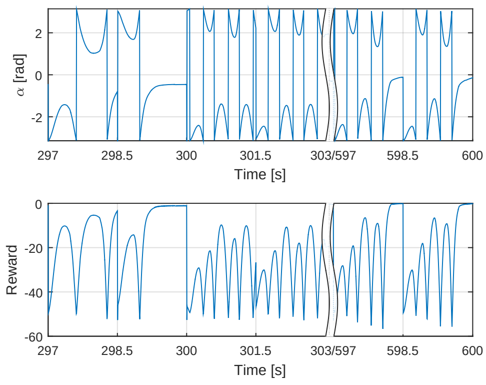


Figure 16: Online LSPI: Time history of the angle and reward, with a jump in g from 9.81 to 19.81 m/s^2 at 300 s, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.05$, using "hanging down ICs", note the gap in the x-axis

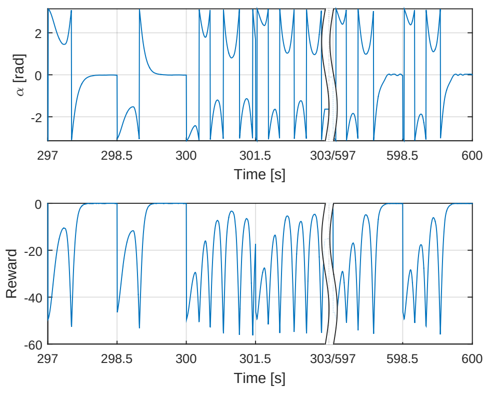


Figure 17: IOLSPI using HD: Time history of the angle and reward, with a jump in g from 9.81 to 19.81 m/s^2 at 300 s, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.05$, using "hanging down ICs", note the gap in the x-axis

5. *Forget factor implementation*

When the conditions change, the set of samples the controller uses to update the policy still largely consists of samples gathered in the old conditions. As such it is expected that when the conditions change, a forget factor η will help. It makes the controller value recent samples more, and forget older ones. After k time steps, a sample will have been reduced to a fraction of η^k of its original value (see algorithm 2). Depending on the forget factor, after a certain number of time steps, the weight of a sample drops below 1 % of its original value and as such is practically discarded. The following equation calculates which forget factor corresponds to a number of samples $N_{0.01}$ that are more than 1% of their original value.

$$\eta = 0.01^{\frac{1}{N_{0.01}}} \tag{17}$$

Three different forget factors are compared: η =1 (no forgetting), $\eta = 0.99991$ (some forgetting, $N_{0.01} =$ 50000 samples) and $\eta = 0.9995$ (much forgetting, $N_{0.01} = 10000$ samples). Figures 18 and 19 show the performance for these different forget factors for online LSPI and IOLSPI using human demonstrations respectively. First of all it can be concluded that a too low forget factor (i.e. much forgetting) yields poor results. This can be explained by looking at the forget factor of 0.9995. With this forget factor only 10000 samples are used, since all samples older than 10000 time steps are practically discarded. It is found that this set of only 10000 samples is not enough to uphold a good value function.

Second, it can be seen that for online LSPI, a forget factor of 0.99991 yields better performance than a forget factor of 1. This can be explained by the fact that the samples at the beginning are forgotten after some time. These samples do not contribute too much, since they are all around the hanging down position. After the change in conditions, the forget factor of 0.99991 also clearly performs better. It forgets the old conditions gradually, and by that allows for a better policy in the new condition.

For the IOLSPI using HDs in figure 19, a little different behavior is seen. With the forget factor of 0.99991, in the first 300 s, the HD samples are being forgotten more and more. This means that the spread in the used samples decreases, and hence the performance also decreases. After the change in conditions, when the HD samples are not of too much interest anymore, since they were obtained under different conditions, the forget factor of 0.99991 shows better results than a forget factor of 1.

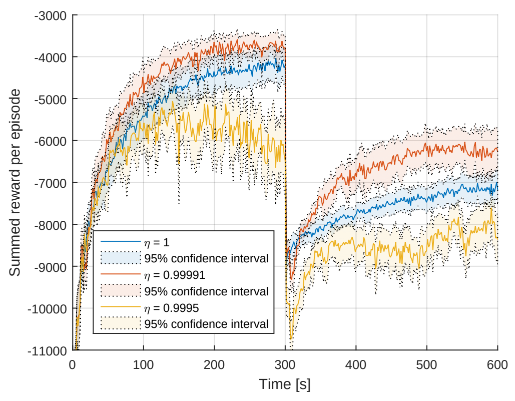


Figure 18: Online LSPI: Summed reward per episode as a function of time for different forget factors, $K_\theta = 1000$, $\epsilon = 0.05$, using "hanging down ICs"

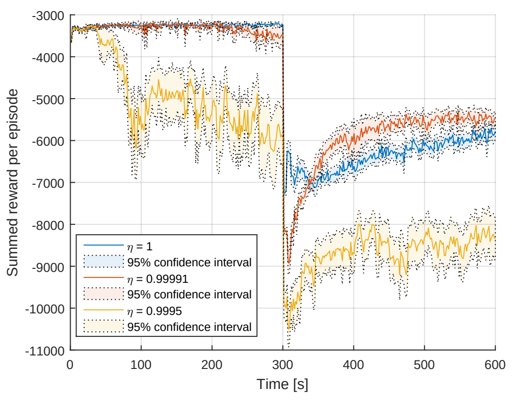


Figure 19: IOLSPI with HD: Summed reward per episode as a function of time for different forget factors, $K_\theta = 1000$, $\epsilon = 0.05$, using "hanging down ICs"

B. Quadrotor slung load system

Having studied the rotational pendulum, the safety aspect in the exploration has not become apparent yet. As such, a second system is studied in which safety plays a more pronounced role. This system is a quadrotor carrying a load suspended from a cable, where the goal is to drop the load in a basket. When the controller is exploring, the quadrotor could either hit a wall or hit its own rotors with the suspension cable. Both events can be considered a crash, with damage to the quadrotor as a consequence. Furthermore, the load could be dropped outside the basket, damaging the (possibly fragile) load. Another interesting aspect of this quadrotor slung load problem is that it has more dimensions than the rotational pendulum. As such the effect of increasing dimension on the performance of the algorithm can be studied.

In this paper, a 2D simplification of the quadrotor system is studied with two degrees of freedom: the lateral position of the quadrotor x, and the swing angle of the load θ. This 2D simplification has more dimensions than the rotational pendulum problem, but still is straightforward to implement. The goal is to drop the slung load at the target as quickly as possible. A snapshot of the simulation is presented in figure 20. Since the quadrotor cannot move up or down, it has to swing the load up to get it in the target basket.

The position of the quadrotor is defined in an inertial frame, with x pointing right and z pointing downwards. The location of the pendulum is described in a radial reference frame, centered in the c.o.m. of the quadrotor. When the pendulum is hanging down, $\theta = 0$ rad. Swinging to the right corresponds to positive θ. The state is defined as $s = [x, \dot{x}, \theta, \dot{\theta}]^T$. The action represents a force exerted on the quadrotor which induces an acceleration $a \in \{-3, 0, 3\}$ m/s^2. A fourth action is present, which is releasing the load.

Considering the dynamics of the system, first the motion of the quadrotor is considered. The influence of the slung load on the quadrotor is neglected. The control input for the system is the acceleration of the quadrotor. It is assumed that no external forces act on the quadrotor, so it can be assumed that $\ddot{x} = a$. The angular acceleration of the slung load is calculated using Eq. (18).[3]

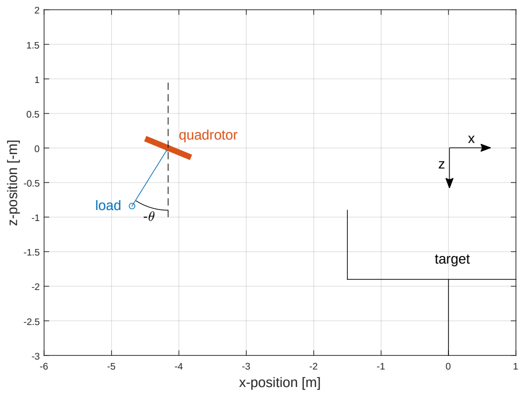


Figure 20: Snapshot of the 2D quadrotor simulation

$$\ddot{\theta} = \begin{bmatrix} -\cos\theta & \frac{\sin\theta}{L} \end{bmatrix} \cdot \begin{bmatrix} \ddot{x} \\ -g \end{bmatrix} \tag{18}$$

In this equation L is the length of the suspension cable. The state is propagated every time step using a simple Euler integration scheme with sampling time T_s:

$$s \leftarrow s + \dot{s} \cdot T_s \tag{19}$$

A number of assumptions are made concerning the slung load. The cable is assumed to be inelastic and massless. Furthermore, it is assumed that the cable is suspended from the c.o.m. of the quadrotor. The system is assumed to be undamped, so aerodynamic forces are neglected, as well as friction of the cable in the suspension point. Finally, it should be noted that the cable is actually modeled as a rod, since the simulation does not allow the cable to become slack.
There are three conditions under which an episode ends:

- The maximum simulation time T_{end} is reached
- A crash occurs (either the quadrotor hits the wall or the cable hits the rotor)
- The load is dropped (either a miss or a dunk)

To score a goal, the following conditions should be fulfilled at the same time:

- The x position of the load is within $x_{goal} \pm x_{g_{margin}}$ where x_{goal} is the basket center, and $x_{g_{margin}}$ defines its width.
- The z position of the load is higher than z_{goal} but lower than $z_{goal} + z_{margin}$
- The load is moving downwards ($\dot{z}_l \geq 0$)

In short, the load must fall in the basket from the top. In table 2 the constants used in this simulation are presented.

Table 2: Dimensions and constants for the quadrotor simulation

Variable	Value	Unit	Description
g	9.81	m/s^2	gravitational acceleration
L	1	m	Cable length
T_s	0.1	s	Sampling time
T_{end}	5	s	Simulation time
z_{start}	0	m	Starting position
x_g	0	m	Goal position
z_g	0.9	m	Goal position
$x_{g_{margin}}$	1.5	m	Goal width

A reward is given when the load is dropped in the goal and a penalty is given when the goal is missed. A slung angle of higher than 90 deg is not desirable, because this would cause the cable to tangle up with the propellers. Therefore, a too high slung angle is penalized in the reward structure. The sides of the domain are represented by walls, so when the quadrotor leaves the domain (it hits the wall) a penalty is given. Furthermore, a penalty is given depending on the distance to the goal, in order to stimulate fast progression to the goal. The reward structure is formalized in Eq. (20).

$$r = \begin{cases} 100 \text{ if dropped in basket} \\ -10 \text{ if dropped next to basket} \\ -50 \text{ if } \theta > 90 \text{ deg} \\ -150 \text{ if quadrotor hits a wall} \\ -0.5 \cdot ||x - x_{goal}|| \end{cases} \tag{20}$$

During learning, the initial conditions are picked at random from the following subsets: $x \in [-6, 0]$ m, $\dot{x} = 0$ m/s, $\theta \in [\frac{-\pi}{3}, \frac{\pi}{3}]$ rad and $\dot{\theta} = 0$ rad/s. This rather distributed spread of initial conditions is chosen to speed up the learning. This makes sure that the spread of the states visited will naturally be a lot bigger than when the initial conditions are always around the same point in the state space.

Similar to the rotational pendulum problem, results are repeated for fifty different random seeds, to account for the randomness. To approximate the Q-function, an equidistant grid of radial basis functions is used. The issue encountered in a system with a higher dimensional state is that the number of basis functions using this an equidistant grid grows exponentially with the number of state dimensions. To make the problem solvable from a computational point of view, the number of RBF centers is limited to the x dimension and the θ dimension, since the velocity and angular rate are believed to be less critical for this problem. For this problem this means that an equidistant grid of 9x1x9x1 RBF centers is used. Since there are four possible actions, a total of 324 RBFs is used.

1. *Results for the quadrotor slung load problem*

Figure 21 shows the learning effect for the quadrotor slung load problem for IOLSPI with a human demonstration and for online LSPI. It can be seen that the performance of IOLSPI is for the first 15000 episodes better than online LSPI. From a safety point of view, figure 22 is especially interesting. It shows the percentage of crashes, misses, and the number of dunks (dropping the load in the goal), as a function of the number of elapsed episodes. It can be seen that the longer the non-initialized LSPI algorithm learns, the lower the amount of misses and the higher the amount of dunks. This is a result of the policy which is improving, and the exploration rate which decays exponentially. The amount of crashes drops in the beginning, and from 10000 episodes onward shows a gradually increasing behavior. This increase in crashes can be attributed to the decreasing exploration: While the exploration is still high, the chance of accidentally dropping the load in the beginning of an episode is large. This dropping of the load ends the episode before a crash could have occurred. After some time, when the exploration rate is lower, the load will remain attached longer, increasing the chance of hitting a wall. It can be concluded that the policy found by the non-initialized LSPI succeeds in reducing the number of misses, but fails to keep the number of crashes low. When human demonstrations are used, it is seen that the performance in terms of dunks and misses is already better from the beginning. The number of crashes is also lower when using human demonstrations. Combining the reduction in misses and crashes over the whole time span, it can be concluded that using IOLSPI with human demonstrations is safer than the non-initialized online LSPI.

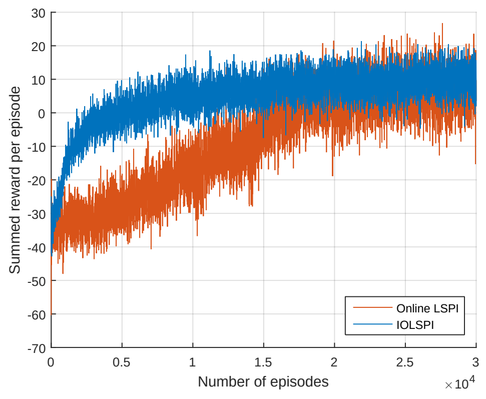


Figure 21: Summed reward per episode as a function of number of episodes for the quadrotor slung load problem

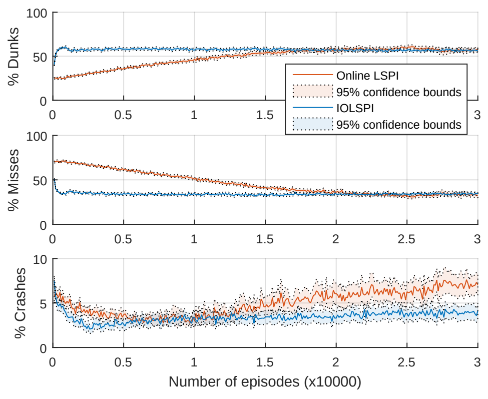


Figure 22: Safety aspects: Percentage of crashes, misses and dunks as a function of number of episodes

For this higher dimensional problem, the execution time becomes a big issue. The time to complete 1000 episodes is compared for different numbers of radial basis functions. Table 3 shows the results. It can be seen that for higher numbers of RBF centers, the computational effort goes up quickly. This can be explained by the fact that within the algorithm, some vector multiplications are done with the vector of basis functions and also with the Γ and Λ matrices, which are square matrices with dimension n (n is the number of basis functions). In the policy improvement step, a linear system of order n is solved which has $O(n^3)$ complexity. This calculation dominates the execution time and explains the strong increase in execution time with increasing number of RBFs.

Table 3: Execution time for different number of RBFs

Total RBFs	x-dimension	$\dot{x}$-dimension	θ-dimension	$\dot{\theta}$-dimension	Computation time [s]
36	9	1	1	1	10.2
324	9	1	9	1	18.5
2916	9	9	9	1	349.5
26244	9	9	9	9	Size of matrices causes memory problems [*]

[*] Computations were executed in Matlab R2015a, which reaches its memory limits.

V. Conclusions and discussion

This paper shows that for some systems, human demonstrations can improve the performance of reinforcement learning in terms of learning time and safety. To show this, a combination of two state of the art reinforcement learning (RL) algorithms is used. Offline Least Squares Policy Iteration (LSPI) and online LSPI are combined to an algorithm named Informed Online Least Squares Policy Iteration (IOLSPI). This algorithm is able to use human demonstration sample sets to improve its performance. The IOLSPI algorithm uses the information present in a batch of samples, which can be obtained by a human demonstration, to kick start the online LSPI algorithm, which continues learning online.

The example studied in this paper is a rotational pendulum problem, where a mass has to be swung up from a hanging down position. It is found that using human demonstrations, a good policy is found straight away, whereas non-initialized online LSPI needs 60000 time steps (300 s) to converge. In addition the final policy found using human demonstrations is 10% better in terms of cumulative reward than the non-initialized online LSPI.

It is found that the benefit of the use of human demonstrations is largest when the human uses its understanding of the problem to efficiently explore the state space, rather than only giving perfect demonstrations of the task at hand. This exploration results in a large variation in samples throughout the state space, which allows for better Q-function approximation. In addition, it is found that human demonstrations work better in IOLSPI when every third action of the human demonstrator is corrupted (replaced by a random action) in order to increase the variety in the demonstration sample set even more.

The benefit of using human demonstration in combination with IOLSPI is larger for problems in which some understanding of the problem is needed to reach a larger portion of the state space. This is the case in the aforementioned swing up pendulum problem. When the rotational pendulum problem is changed in such a way that each episode of the problem is started at a random state, no understanding of the problem is needed to explore the whole state space, in contrast to the original problem, which always starts hanging down. For this modified problem, it is found that the IOLSPI using human demonstrations finds a good policy straight away as well. However, in this modified problem the non-initialized online LSPI algorithm finds a good policy already after 20000 time steps (100s), significantly faster than in the original problem. Furthermore, the found policy scores only 1% worse than the policy found by the IOLSPI algorithm using human demonstrations.

In addition, it is shown that the IOLSPI algorithm fulfills the demands for an autonomous controller: It is fully model-free and it can adapt to changing conditions. To improve the adaptability of the algorithm, a forget factor is implemented, to value outdated samples less than more recently gathered ones, to focus on the current conditions. It is found that a mild forgetting behavior results in better performance in changing conditions. With a mild forget factor, the IOLPSI algorithm needs over 40000 time steps less to obtain a similar performance after a change in conditions, compared to the algorithm without the use of a forget factor.

With a second studied system the benefit in terms of safety of the IOLSPI algorithm is shown. In this quadrotor slung load drop off task it is shown that the number of crashes while learning is reduced by 10% to 50% using the IOLSPI algorithm, compared to a non-initialized reinforcement learning algorithm.

With the second case study, the limitation of the algorithm becomes apparent. A uniformly distributed grid of radial basis functions is used to approximate the Q-function. With a growing number of state parameters, the number of basis functions grows exponentially. For higher dimensional problems, the computational effort becomes too high for present day computers.

In future research, improvements could be made by combining the IOLSPI algorithm with methods that choose the radial basis functions in an intelligent way, or use other basis functions. When the number of basis functions is limited, the computational effort stays low. With this addition in place, the algorithm can be applied to multiple actuator systems, with continuous actions. With these extensions, the algorithm can be applied to a 3D quadrotor simulation, and eventually on a real quadrotor. This will be a next step towards model-free and adaptive control supporting the autonomy of these complex vehicles.

References

[1]Kumar, V. and Michael, N., "Opportunities and challenges with autonomous micro aerial vehicles," *The International Journal of Robotics Research*, Vol. 31, No. 11, 2012, pp. 1279–1291.

[2]de Croon, G. C. H. E., Groen, M. A., De Wagter, C., Remes, B., Ruijsink, R., and van Oudheusden, B. W., "Design, aerodynamics and autonomy of the DelFly," *Bioinspiration & Biomimetics*, Vol. 7, No. 2, 2012.

[3]Faust, A., Palunko, I., Cruz, P., Fierro, R., and Tapia, L., "Learning swing-free trajectories for UAVs with a suspended load," *IEEE International Conference on Robotics and Automation, 2013*, 2013, pp. 4902–4909.

[4]Barto, A. G. and Sutton, R. S., *Reinforcement learning: An introduction*, The MIT Press, Cambridge, Massachusetts, 1998.

[5]Busoniu, L., De Schutter, B., Babuska, R., and Ernst, D., "Exploiting policy knowledge in online least-squares policy iteration: An empirical study," *Automation, Computers, Applied Mathematics*, Vol. 4, 2010, pp. 521–529.

[6]Vaandrager, M., Babuska, R., Busoniu, L., and Lopes, G., "Imitation learning with non-parametric regression," *IEEE International Conference on on Automation, Quality and Testing, Robotics, 2012*, 2012, pp. 91–96.

[7]Abbeel, P., Coates, A., and Ng, A. Y., "Autonomous Helicopter Aerobatics through Apprenticeship Learning," *The International Journal of Robotics Research*, Vol. 29, No. 13, 2010, pp. 1608–1639.

[8]Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D., *Reinforcement learning and dynamic programming using function approximators*, CRC Press, Boca Raton, Florida, 2010.

[9]Lagoudakis, M. G., "Least-squares policy iteration," *Journal of Machine Learning Research*, Vol. 4, 2003, pp. 1107–1149.

[10]Busoniu, L., Ernst, D., De Schutter, B., and Babuska, R., "Online least-squares policy iteration for reinforcement learning control," *American Control Conference , 2010*, 2010, pp. 486–491.

[11]Palunko, I., Faust, A., Cruz, P., Tapia, L., and Fierro, R., "A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots," *IEEE International Conference on Robotics and Automation, 2013*, 2013, pp. 4896–4901.

[12]Palunko, I., Donner, P., Buss, M., and Hirche, S., "Cooperative suspended object manipulation using reinforcement learning and energy-based control," *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 885–861.

[13]Faust, A., Palunko, I., Cruz, P., Fierro, R., and Tapia, L., "Automated aerial suspended cargo delivery through reinforcement learning," *Artificial Intelligence*, Vol. 1, 2014, pp. 1–18.

[14]Bellman, R. E., *Dynamic Programming*, Princeton University Press, New Jersey, 1957.

[15]Bertsekas, D. P., *Dynamic Programming and optimal control 3rd ed.*, Athena Scientific, Bellmont, Massachusetts, 2011.

[16]Li, L., , Littman, M. L., and Mansley, C. R., "Online exploration in least-squares policy iteration," *International Conference on Autonomous Agents and Multiagent Systems, 2009*, Vol. 2, 2009, pp. 733–739.

Part II

Preliminary Research

Chapter 2

Introduction to the Preliminary Research

Micro Aerial Vehicles (MAVs) have shown a great potential in many applications due to their low cost and high maneuverability [1]. Transport of small cargoes, remote sensing and search and rescue missions are tasks for which MAVs are well suited. Especially in tasks at locations which are hard to reach for humans, e.g., for visual inspection of offshore wind turbines, MAVs can play an important role. Nowadays, quadrotors are the most used MAV platforms for these applications because of their simple geometry and agility [1]. Human pilots are able to fly quadrotors rather well, but making them fly autonomously saves time and eventually could improve mission performance. In low level control tasks such as attitude hold, modern control algorithms outperform human expert pilots. Furthermore, for tasks in which the MAV is out of the line of sight of the human pilot and could lose contact with him or her, autonomous flying capability is necessary for the MAV to complete the task.

Enabling autonomous flight is difficult because of the complex dynamics of quadrotors and the potentially changing conditions in which they fly. At first, the controller should be adaptive in order to keep up with the changing environment. Secondly, the controller should be model free. Since the dynamics of quadrotors are complex, using a model based controller inevitably results in modeling errors. Model free controllers do not have this problem and are more easily extendable to even more complex MAV platforms, such as the Delfly [2].

Reinforcement Learning (RL) is an an approach which can both adapt to changing conditions and is able to function with no prior knowledge of the dynamics of the system [4]. In RL an agent (controller) interacts with its environment collecting rewards for good behavior and penalties for bad behavior. In the beginning the policy is not perfect yet, so the agent tries actions which explore the unknown part of the state space. The agent uses the rewards to update its policy (control strategy) in order to maximize the expected cumulative reward. A downside of using RL in quadrotor control is that because of the large state and action space for a quadrotor, training towards a good policy takes a lot of time and will probably lead to crashes while the controller is still learning a good policy.

This training time might be reduced considerably by the use of a-priori knowledge within the controller. A way of doing this is the use of a skilled "teacher" which demonstrates a maneuver [8]. This kind of supervised learning could be a starting point for the reinforcement controller which uses these demonstrations as a basis for its policy.

The goal of this research is to find out in which way demonstrations of a human expert pilot can be used to improve the performance of reinforcement learning for quadrotor control. To achieve this goal, the following research question will be answered. *How can we use demonstrations of a human expert pilot to improve the performance of reinforcement learning for quadrotor control?* This research question is subdivided in the following sub-questions:

- How can human expert demonstrations be used within RL algorithms?
- How can the performance of a RL algorithm be defined?
- How do RL algorithms using human expert demonstrations compare to each other, and how do they compare to existing techniques?

The challenge in this research endeavor will be threefold. Firstly, the algorithm has two opposing goals. On the one hand the algorithm should be generally applicable, on the other hand it should be able to quickly learn a policy. Secondly, there will be a challenge in defining a good metric to assess the performance of different algorithms, in terms of their result on a task, required training time, etc. Finally, there will be a challenge in finding a good way to use the human demonstrations. Since there will be significant variance, and probably also failure in these demonstrations, it could be difficult to extract the right information.

To make the problem more tangible and to have a standard to compare the performances of the different methods, a benchmark task is selected. This benchmark task will be a slung load drop-off task; the quadrotor starts from an arbitrary position with respect to the drop-off target and has to drop off the slung load as quickly as possible. This task was selected for two reasons:

1. Because of the slung load, the dynamics are even more complex than for a quadrotor alone, which makes the problem, in combination with the non trivial task, suited to apply reinforcement learning.
2. The task is difficult, but doable for a human; an expert human pilot should be able to achieve a good performance on this task and at least drop the slung load at its goal in a percentage of his/her trials.

This preliminary research part is structured as follows. In Chapter 3, some basic concepts and notations on RL are presented. Chapter 4 gives an overview of current literature on RL, with a focus on RL in quadrotor applications. Chapter 5 will provide a simple simulation in which two RL algorithms will be tested. Chapter 6 describes the approach taken in the remainder of the research and gives a brief overview on what RL algorithms will be used in the further analysis. Furthermore, it gives an outline of the intended 3D simulation and experiment. Finally, Chapter 7 will provide the conclusions on this preliminary research.

Chapter 3

Preliminaries on Reinforcement Learning

This chapter presents the necessary basic knowledge on reinforcement learning theory that is used in the remainder of this preliminary research. Throughout literature, many different notations are used within reinforcement learning algorithm mathematics. As such this chapter serves as an overview of the used notation and terminology within this report. The reinforcement learning problem describes a decision process in which decisions or actions result in, possibly delayed, rewards. Figure 3-1 shows the general idea in reinforcement learning. The agent/controller decides on an action to take in the environment. The environment will end up in a new state and give a reward. Using this information, the agent updates its policy and decides on a new action. The goal within the reinforcement learning for the agent is to develop a policy which maximizes the accumulated reward collected.

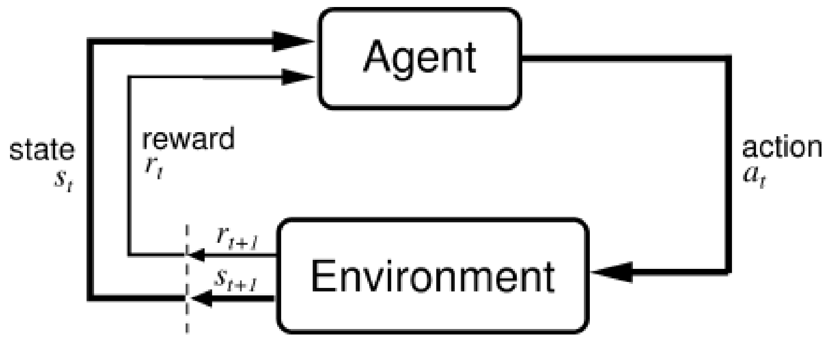


Figure 3-1: The agent-environment interaction in reinforcement learning [4]

3-1 Markov Decision Process

In general, reinforcement learning algorithms solve a Markov Decision Process (MDP). An MDP provides a framework for modeling problems involving sequential decision making. Formally, a deterministic MDP is defined by a state space S, which contains all possible states s ($s \in S$), an action space A, which contains all possible actions a ($a \in A$), a transition function $f(s, a)$ and a reward function $\rho(s, a)$ [9]. The transition function is a representation of the behavior of a system, i.e., it determines in what state the system ends up after applying an action (3-1).

$$s_{k+1} = f(s_k, a_k) \tag{3-1}$$

Where $k = \{1, 2, \ldots, K\}$ and K is the total number of discrete time steps. The reward function calculates the immediate experienced reward following performing a certain action a in state s (see Eq. 3-2). The reward gets index $k+1$ since it follows after the current state and action, and is an input for the agent in the next time step.

$$r_{k+1} = \rho(s_k, a_k) \tag{3-2}$$

The Markovian property states that the transition model and reward function do not change as function of the history of states visited, which actually means that when one observes the current state and possible actions, there is no need to know what happened before that state to make a decision on which action to take [4].

The goal within an MDP is to maximize the return. The return is the accumulated reward, which is calculated using 3-3.

$$R^{\pi}(s_0) = \sum_{k=0}^{T} \gamma^k \rho(s_k, \pi(s_k)) \tag{3-3}$$

In this equation, γ is the discount factor, which has a value between 0 and 1. When this factor is lower than one, immediate rewards are valued higher than rewards in the future. In an MDP the agent selects an action a according to its policy π. This policy is a mapping from states to actions and determines the performance of the controller given the characteristics of the MDP.

An MDP can last for an infinite timespan, or it can be episodic. This means that there is a start state and an end state; the number of time steps is finite. Each realization, from start state to end state, is called an episode.

A quadrotor flying with a slung load can be considered a Markov Decision Process [10]. There is a state space S, which consists of all possible combinations of location, velocities etc. for the quadrotor system. Furthermore, a number of actions can be performed by the quadrotor, e.g., pitching forward, increase thrust etc. A reward function can be defined in order to reach a certain goal, and the transition function is given by the dynamics of the system, which might be unknown. There is one aspect of a quadrotor in real flight that may violate the Markovian property; since the inflow on the rotors affects the thrust of the rotors, but is not included in the state, since this is difficult to measure, the same set of states could result in different behavior, depending on previous states. The implications of this are, however, expected to be small. As a result, as is done in the reviewed literature, it is assumed that this effect will not cause significant problems for this research.

3-2 Value functions

As mentioned before, the goal in an MDP is to find an optimal policy which maximizes the experienced return. This return is not known beforehand, so value functions are used as a measure of the expected return of a certain state. We distinguish between state values and state-action values. The state value (V-value) is defined by 3-4 [9] and represents the expected return following policy π, starting from state s.

$$V^{\pi}(s) = E\{R^{\pi}(s)\} \tag{3-4}$$

The state-action value function (Q-value) is defined as the return obtained, while at state s, taking action a, and after this following policy π (3-5).

$$Q^{\pi}(s,a) = E\{R^{\pi}(s)|a\} \tag{3-5}$$

For small and discrete state and actions spaces, the value for each state and action can be quickly updated and used to evaluate which action should be chosen, to end up in in the next state with the highest value. When the number of states and actions grows however, the number of states for which a value should be estimated explodes, this is called the curse of dimensionality [4]. For a quadrotor for example, the state is not discrete, meaning that there is an infinite number of states. As a result, it becomes impossible to assign a value to each state individually.

To solve this problem, the value of a state can also be approximated globally. This can be done using for example a neural network, but a more straightforward way is to use basis functions. These basis functions ($\phi_l(s)$ with $l = 1 \dots n$) are multiplied with a weight (w_l), and summed to get an approximation of the value of a certain state, or state-action pair. For approximate V-functions equation 3-6 holds, and for approximate Q-functions equation 3-7 holds.

$$\hat{V}(s) = \sum_{l=1}^{n} w_l \phi_l(s) = W^T \cdot \Phi(s) \tag{3-6}$$

$$\hat{Q}(s,a) = \sum_{l=1}^{n} w_l \phi_l(s,a) = W^T \cdot \Phi(s,a) \tag{3-7}$$

These basis functions can have any form. They can be general functions, such as Chebyshev polynomials, or task specific to include some knowledge on the task and system in the value approximation. A more elaborate discussion on value functions will follow in Chapter 3.

3-3 Methods for solving MDPs

The goal in solving the MDP is to find the policy that maximizes the return $R(s_0)$. A multitude of methods exist to do so [4]. Usually, a distinction is made between model based

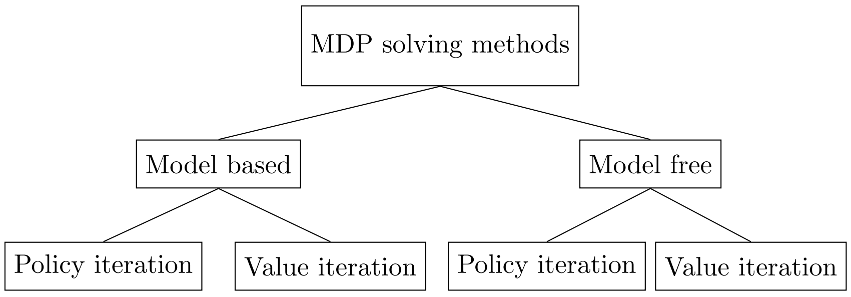


Figure 3-2: Taxonomy of MDP solving methods

and model free approaches. In model based approaches, the transition function $f(s, a)$ is known or an approximation of it is available. For a quadrotor, when a dynamic model of the vehicle is used, the algorithm is model based. Problems in which a transition model is available, can be solved using Dynamic Programming (DP) techniques. In model free algorithms, there is no knowledge on the transition function. When there is no information on beforehand at all, the agent needs to find the optimal policy by just interacting with its environment and experience which actions are better and worse.

Furthermore, a distinction is made between online and offline algorithms. Offline algorithms learn a policy using a model, before performing the task in real life. Online algorithms on the other hand, learn a policy while performing the real task, "on the fly". There are algorithms which combine both; they use a simulator to offline learn an initial policy, but online adjust the policy "on the fly". In online algorithms there is the need to explore, i.e., the agent should every now and then take an action which is the one that, at the current knowledge of the agent, does not provide the highest return. If the agent would only exploit its current knowledge, it would never find out whether there are states which actually have higher value than the states it is exploiting. A common and straightforward way of implementing exploration in an algorithm, is to include randomness factor ϵ which has a value between zero and one. In each state, with a probability of ϵ, a random action is chosen.

Whether the algorithm is model based or model free, online or offline, its goal is to maximize the return, doing this by updating the policy it is following. For the update of the policy there are two general methods which are used for either model based or model free methods: policy iteration and value iteration. Graphically, the connection between these methods is presented in Figure 3-2.

3-3-1 Policy iteration

In policy iteration the algorithm iterates between two steps. The first step is policy evaluation, which means that the value of each state, using the current policy, is calculated [11], this can be seen in line 3 of Algorithm 1. The second step is policy improvement, which means that a new policy generated which is greedy towards the current value function. Policy iteration is called on-policy because in policy iteration, the policy which is used to control the process is also the policy that is evaluated [4]. A basic version of policy iteration is SARSA, which is the algorithm described in Algorithm 1. It is called SARSA because it uses the State, Action, Reward, (next) State, (next) Action, to update the value estimate and thereby indirectly the policy.

Algorithm 1 Policy iteration basis (SARSA) [11]

```
1: for every iteration $k = 1, 2, ...$ do
2:     Apply $a_k = \pi_k(s_k)$, measure next state $s_{k+1}$ and reward $r_{k+1}$
3:     Next action using current policy would be $a_{k+1} = \pi_k(s_{k+1})$ (on-policy)
4:     Update Q: $Q_{k+1}(x_k, a_k) \leftarrow Q_k(x_k, a_k) + \alpha_k[r_{k+1} + \gamma Q_k(x_{k+1}, a_{k+1}) - Q_k(x_k, a_k)]$
5: end for
```

3-3-2 Value iteration

In value iteration the algorithm looks for the optimal value function, that is for the maximum return for a state or state-action pair. A straightforward example of value iteration is Q-learning (Algorithm 2). The optimal value function is used to find the (optimal) policy. In value iteration algorithm the value is determined looking at the maximum value of the next state (Algorithm 2, line 3), instead of the value of the next state using the current policy, as policy iteration does. As such, the policy according to which it acts, is not the same as the policy that is evaluated. In practice this means that in the policy there is some exploration, whereas in the evaluation there is not, as such this is called an off-policy algorithm [4].

Algorithm 2 Value iteration basis (Q-learning) [11]

```
1: for every iteration $k = 1, 2, ...$ do
2:     Apply $a_k = \pi_k(s_k)$, measure next state $s_{k+1}$ and reward $r_{k+1}$
3:     Update Q: $Q_{k+1}(s_k, a_k) \leftarrow Q_k(s_k, a_k) + \alpha_k[r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k)]$
4:     In which is should be stated that $Q_{k+1}$ is evaluated using the maximum value for the
   next state (Off-Policy)
5: end for
```

3-4 Overview

Concluding this background on reinforcement learning basics, the quadrotor slung load task can be considered to be a good problem to study from a reinforcement learning perspective. The process can be considered an episodic Markov Decision Process for which an optimal policy will be found using a reinforcement learning method. By nature, the process is continuous in both state and action, and since a slung load drop off task is considered, there is a clear end of the task, as such it is episodic. Whether the used algorithm will work based on policy iteration or value iteration, has to be found out within the literature study. A more elaborate introduction on reinforcement learning concepts can be found in the book of Sutton and Barto [4] and the work of Wiering and Otterlo [9]. For a thorough introduction on continuous state reinforcement learning theory the book of Busoniu [11] provides a good overview.

Chapter 4

Literature Review

In order to determine how human demonstrations can be used to improve reinforcement learning in quadrotor control, first a literature study is conducted. In this literature study, the goal is to find out what reinforcement learning algorithms have been used to date within quadrotor control. Furthermore, it is investigated in which ways human expert demonstrations have been used within these algorithms. Finally, the way in which the performance of these algorithms is evaluated is studied. Within this study, both literature on simulations of quadrotor tasks, as well as literature describing real experiments, are reviewed. Since the reference task selected in this research is the slung load drop off task, a significant portion of the literature studied is on slung load related tasks.

Within the methods found in literature, a distinction has been made between model based and model free methods. Some methods described could be placed in either of the categories. In this research the distinction is made as follows: when an algorithm uses a transition model in the learning of a new policy, it is considered model based, otherwise it is considered model free. For example, in Continuous Action Fitted Value Iteration (CAFVI); in the actual learning of the policy no model is used, however, some dynamics samples are used within the algorithm, which could both be obtained in advance by using a simulator or human demonstrations. This example thus falls under the category model free. Using this distinction between model based and model free algorithms, Figure 4-1 has been composed to show different algorithms used within quadrotor control, studied in this literature survey. In the gray boxes are the algorithms as applied in the corresponding papers, and the section in which they are discussed.

This chapter is structured as follows. In section 4-1, the different components of which reinforcement learning algorithms consist within quadrotor control, are elaborated on. After this, some full algorithms are studied in section 4-2 (model based) and section 4-3 (model free). In section 4-4 different performance metrics are discussed. Finally, section 4-5 provides the conclusions on the literature study.

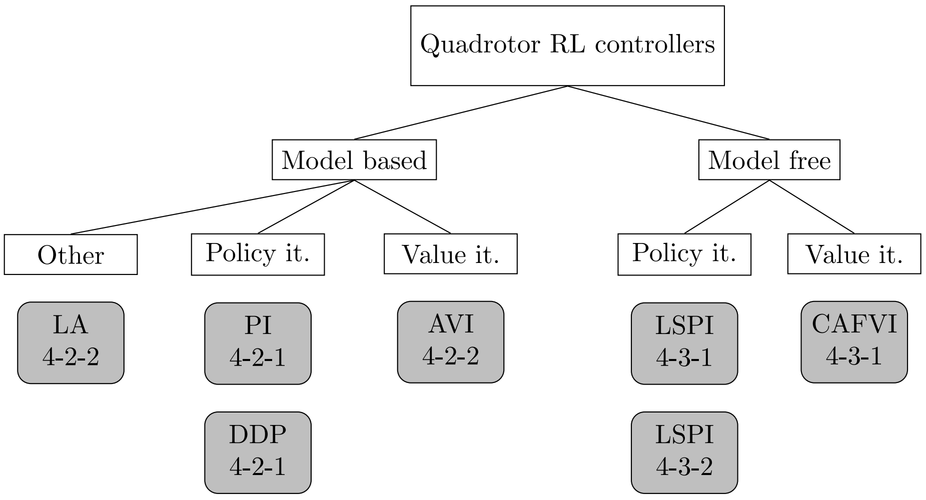


Figure 4-1: Taxonomy of different reinforcement learning methods used for quadrotor control with the sections in which they are discussed. *(LA: Learning automata, PI: Policy iteration, AVI: Approximate Value Iteration, LSPI: Least Squares Policy Iteration, CAFVI: Continuous Action Fitted Value Iteration, DDP: Differential Dynamic Programming)*

4-1 Components of a reinforcement learning algorithm

Studying the literature of reinforcement learning algorithms that have been used for quadrotor control tasks, a number of components can be identified that constitute such algorithms. Figure 4-2 shows these main components. The five components on the left of the figure, constituting a reinforcement learning algorithm, are distinguished: A reward function, value basis functions, a transition function, a policy, and a way of updating that policy. To the right of each big block are the options listed for that component. Each RL algorithm studied is a combination of these options. It must be stated, however, that not always all components are present. For example, in model free RL a transition function is not used. In the following sections the different components will be described in more detail.

4-1-1 Reward functions

The reward function specifies what behavior gets rewarded and thereby implicitly defines the goal of the process [9]. The reward function can be open or regulative. An open reward function only rewards when the final goal is reached, leaving the way to the goal open. A regulative reward function rewards intermediate steps that lead towards the final goal as well, thereby regulating the agent more towards the goal.

In quadrotor literature, different degrees between open and regulative are identified. In the work of Abbeel [8], a highly regulative reward function is used to fly acrobatic maneuvers. 24 features are used in the reward function. This resulted in a good performance on this task, with the behavior prescribed rather strictly with this approach. To use such a regulative reward function, one should have a good idea of a proper strategy in advance. The result of

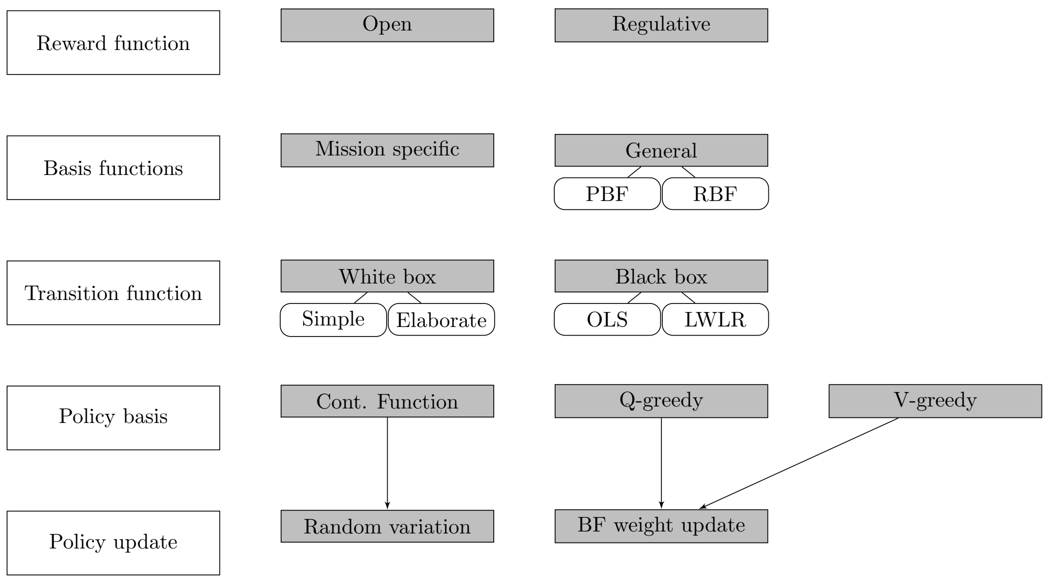


Figure 4-2: General set-up of a reinforcement learning algorithm with on the left in white the components, and on the right the possible options to use in these components. *PBF: Polynomial Basis Functions, RBF: Radial Basis Functions, OLS: Ordinary Least Squares, LWLR: Locally Weighted Linear Regression*

this is that the reinforcement learning algorithm does not have a lot of freedom to find other strategies.

A rather open reward function is used by Figueroa [12] for the balancing of an inverted pendulum on a quadrotor. The reward function only rewards the final goal, which is a straight up pendulum, and for all other situations gives zero reward. This task is non minimum phase with no trivial solution. The openness does allow for a lot of freedom of the algorithm, making it possible to find an upswing strategy to balance the inverted pendulum. A downside of this approach is that it takes more iterations to reach a good policy than for a regulative reward function, since the agent only knows when it is perfectly good, but has no idea when it is close to a good state.

An intermediate case between these is used by Bou-Ammar [13], pursuing a hover task. Here the offset of the attitude angles are in the reward function, which increases reward towards the final goal. This means that, in contrast to using the reward function of Figueroa [12], the agent can tell when it is near the final goal and not only when it is spot on.

In general, it is found out that most used reward functions in literature are continuous as function of for example the offset from a goal state similar to the example of Bou-Ammar [13]. Another finding is that in many applications, the control action is penalized in the reward function, to limit control effort of the quadrotor.

4-1-2 Basis functions for value approximation

The second component is the used set of basis functions to estimate the value of states, or state-action pairs. In quadrotor literature, two types of basis functions are distinguished: mission specific basis functions and general basis functions. Mission specific basis functions incorporate knowledge on the system and task at hand, to select some specific basis functions. General basis functions use an arbitrary combination of state and action variables. Bou-Ammar [13] and Faust [14] use a mission specific set of basis functions. The knowledge on the problem can be used to suffice with less basis functions in estimating the value. The task to be performed by the quadrotor discussed by Bou-Ammar is hover stabilization. As such the basis functions are chosen to be the squares of the attitude angles (ϕ, θ and ψ) and some coupling terms as in Equation 4-1.

$$V(s) = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} \cdot \begin{bmatrix} \phi^2 \\ \theta^2 \\ \psi^2 \\ \phi\psi \\ \theta\psi \end{bmatrix} \tag{4-1}$$

With this approach, the value is clearly coupled to the task at hand. As a result, the use of these basis function can not be extended to different tasks.

The other option is to use general basis functions to estimate the value for states or state-value pairs. In literature, polynomial basis functions and radial basis functions are used. Shaker [15] uses both methods and shows that for a landing task they yield similar results. In the work of Palunko ([10] and [16]), polynomial basis functions are used, in this case Chebyshev polynomials from the first kind (Equation 4-2). Figure 4-3 shows these polynomials.

$$\begin{aligned} \psi_0(\bar{\xi}) &= 1 \\ \psi_1(\bar{\xi}) &= \bar{\xi} \\ \psi_{j+1}(\bar{\xi}) &= 2\bar{\xi}\psi_j(\bar{\xi}) - \psi_{j-1}(\bar{\xi}) \end{aligned} \tag{4-2}$$

To use these Chebyshev polynomials ($\psi_n | n = 0, 1, 2, \ldots N$), the state and action variables should be normalized (Equation 4-3).

$$\bar{\xi} = -1 + 2\frac{\xi - \xi_L}{\xi_H - \xi_L} \tag{4-3}$$

In these equations ξ is the state variable that is used to build a Chebyshev polynomial with. $\bar{\xi}$ is the normalized Chebyshev variable, and indices H and L denote the highest and lowest value of the parameter respectively.

Using these Chebyshev polynomials, the set of basis functions for the whole state-action space is given by Equation 4-4.

$$\phi(x, u) = \Phi(u) \otimes \Psi(x) \tag{4-4}$$

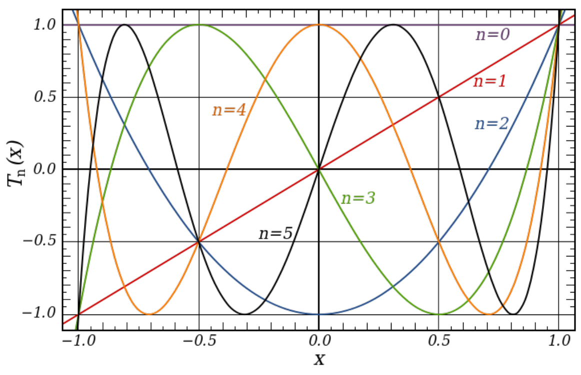


Figure 4-3: Graphical representation of Chebyshev polynomials of the first kind by Whillas [17].

With $\Phi(u)$ and $\Psi(x)$ are the vectors of Chebyshev polynomials for the action and state respectively. $\otimes$ is the Kronecker product [10], so 4-4 can be rewritten to 4-5.

$$\begin{bmatrix} \phi_1(x,u) \\ \phi_2(x,u) \\ \vdots \\ \phi_M(x,u) \\ \phi_{M+1}(x,u) \\ \vdots \\ \phi_{NM} \end{bmatrix} = \begin{bmatrix} \Phi_1(u)\Psi_1(x) \\ \Phi_1(u)\Psi_2(x) \\ \vdots \\ \Phi_1(u)\Psi_M(x) \\ \Phi_2(u)\Psi_1(x) \\ \vdots \\ \Phi_N(u)\Psi_M(x) \end{bmatrix} \tag{4-5}$$

The advantage of using these general basis functions, is that no knowledge on the task and vehicle need to be included, which makes the approach flexible towards other tasks and vehicles. The downside of the Chebyshev polynomial approach, is that it does not work so well for multiple input systems. However, many polynomial or radial basis function methods exist to avoid this problem. Generally, literature shows that mission specific basis functions are the most common used approach. It was found out however, that general basis functions can also result in well performing algorithms.

4-1-3 Transition functions

As mentioned in Chapter refch:rl, model based algorithms use a transition function which maps a state-action combination to a new state. Within quadrotor reinforcement learning literature, two ways of finding a transition function are identified. Either a model is made describing the dynamics of the system (white box approach), in order to calculate the new state, or an input-output model is derived from a training data set (black box approach). The latter can be considered as supervised learning [18].

Among the white box approaches, different levels of detail are discerned. Some methods only use a simple model of the dynamics of the vehicle, and as such use it only as a rough estimate of the next state. Faust uses this approach in [14] and [3]. Both papers elaborate on a slung load delivery task, in which the dynamic model is a simple integration scheme to estimate the next state. Equation 4-6 shows the calculation by Faust [3] for the velocity (v), location (p), and slung load angle (η). Here the input is the acceleration a, and is slung load angular acceleration $\ddot{\eta}$ calculated using simple trigonometry. This simple model is only used as a rough estimate of the new state following from certain inputs. The obtained estimated new states are then used to evaluate their respective value, to choose the best action.

$$v = v_0 + \Delta t a \quad p = p_0 + \Delta v_0 + \frac{\Delta t^2}{2} a \tag{4-6}$$

$$\dot{\eta} = \dot{\eta}_0 + \Delta t \ddot{\eta} \quad \eta = \eta_0 + \Delta t \dot{\eta}_0 + \frac{\Delta t^2}{2} \ddot{\eta} \tag{4-7}$$

In algorithms using dynamic programming to solve the reinforcement learning problem, some more elaborate models are used. An example of this can be found in a paper of Palunko [19]. Here a more elaborate dynamics model is used, to solve for the optimal policy using dynamic programming.

Instead of composing a white box model of the dynamics of the vehicle, a black box system identification approach could also be taken, in order to be able to estimate the transition function. Many black box supervised learning algorithms exist to make an input-output mapping for a system. Advanced nonlinear methods, such as artificial neural networks and splines, could be used to make an input-output mapping. In quadrotor application only linear regressions are found however. A good example of linear regression is the work of Waslander [20], where locally weighted linear regression is used. This approach extends normal least squares approximation to a weighted least squares approximation. The weights are defined by the distance of each training sample input, to the input for which the output has to be calculated. This weighting is done using Equation 4-8.

$$W_{i,i} = e^{\frac{-||x_i - x||}{2\tau^2}} \tag{4-8}$$

In this equation, x is the input for which the output should be computed, x_i is the i^{th} input of the training set, and τ is a fitting parameter which determines the range of influence of the training samples.

The advantage of this method is that the local effects within the data fitting are included up to an extent that can be adjusted using τ. This advantage makes this method more usable for quadrotor controllers than a least squares approximation with no weighting, since it takes into account local effects.

4-1-4 Policy basis

Within literature on reinforcement learning for quadrotor control, three different bases for a policy are found. A distinction can be made between policies that are greedy towards a state-action value, policies that are greedy towards a state value, and policies that are described as a continuous function of the state.

In the literature reviewed, the majority of the algorithms is greedy towards a state value or a state-action value. Bou-Ammar [13] and Figueroa [12] use a policy which is greedy towards the approximated value. This policy can be formalized as in Equation 4-9.

$$\pi(s) = \arg\max_a V(s) \tag{4-9}$$

In algorithms that use an action-value function approximation in stead of a value function approximation, the policy can be greedy towards the action-value function. In the work of Palunko [10] this approach is used. The policy can be formalized as in Equation 4-10.

$$\pi(s) = \arg\max_a Q(s, a) \tag{4-10}$$

It should be noted that, having a good approximation of the value, the greedy policies described here prove to be successful approaches. When this is not the case however, for example when the algorithm is run online, such as in the effort of Palunko [10], a greedy policy will not suffice, since at the start the approximation of the value might not be good, yielding a sub optimal policy. To this end, instead of a greedy policy, an ϵ- greedy policy can be applied [10]. In that case, with a probability of ϵ, a non-greedy action will be chosen at random. Because of the improvement of the value approximation alongside with the iterations, this ϵ can decrease over time.

The third method is used by Waslander [20], where the policy is described by a function of linear functions of some of the state parameters. The task studied by Waslander is an altitude control task, which tries to follow a reference altitude. The policy is described by Equation 4-11, and as such is a function of the current state and some weights w_1, w_2, w_3, w_4.

$$\pi(s, w) = w_1 + w_2(p_z - p_{z,ref}) + w_3 v_z + w_4 a_z \tag{4-11}$$

Using this approach the policy is a smooth function of the state (position p, velocity v and acceleration a in z-direction). The weights are to be learned by the algorithm, to achieve a good performance.

The advantage of the latter method over the former two, is that there is no need to evaluate the value over all possible actions given a state. Especially when the state is large and/or continuous, this value evaluation can become computationally intensive. On the other hand, the advantage of the former two methods is that for each state they find the one optimal action, which is not guaranteed for the latter method.

4-1-5 Policy update

The key in reinforcement learning is that the policy improves over time. This update can be performed in two ways: Either updating the policy itself (when this is a function of the state), or updating the weights of the value function (when the policy is ϵ-greedy towards the value function). The following sections describe different approaches to update the policy or its value function approximation.

Direct policy update using random variation of policy weights

As described in subsection 4-1-4, in the algorithm described by Waslander [20], the policy is defined as a linear combination of weights and state parameters. As such the policy is updated directly by changing the weights within the policy. The method to update these weights is presented in Algorithm 3.

The method used is to initialize the policy weights (line 3) with reasonable values. From this initial situation, the algorithm evaluates the current policy by looping over a number of initial states, and within that over a whole episode. The total reward obtained in such a loop is a measure of the performance of the policy. The policy is evaluated (line 7-14); if the performance using the current policy is the new best, the policy weights are stored (line 16). After this, the policy is updated by adding a random Gaussian noise vector to the policy weights (line 18), and the cycle starts over again.

This policy update method is model based; it uses a transition function to calculate the next state (line 11). The method seems useful, since it is expected to converge to the optimal policy by the use of adding Gaussian random numbers to the weights. A downside of the method is its dependence on the initialization of the policy weights: if they are far off the optimal policy weights, it will take a long time to converge to the optimal policy. Furthermore, a gradient method might be faster than this random approach, because such a method is more directed towards the optimal policy than this random approach. The method used here does have the advantage that if the weight updates are big enough, it is able to escape from local minima.

Algorithm 3 Policy update by adding Gaussian noise [20]

```
1: Generate set $S_0$ of random initial states
2: Generate set $T$ of random reference trajectories
3: Initialize $w$ with reasonable initial guess
4: Initialize $v$ which is a noise realization
5: $R_{best} = -\infty$, $w_{best} = w$
6: repeat
7:     $R_{total} = 0$
8:     for $s_0 \in S_0, k \in T$ do
9:         $s(0) = s_0$
10:        for $k = \{0, 1, 2, \dots, K-1\}$ do
11:            $a_k = \pi(s_k, w)$
12:            $s_{k+1} = f(s_k, a_k) + disturbance$
13:            $R_{total} = R_{total} + \rho(s_{k+1})$
14:        end for
15:    end for
16:    if $R_{total} > R_{best}$ then
17:        $R_{best} = R_{total}, w_{best} = w$
18:    end if
19:    Add Gaussian random vector to $w_{best}$, store as $w$
20: until $w_{best}$ converges
```

Least squares policy iteration

As mentioned, most of the policy updates are done not directly but via the update of the value function instead. An option to do this is called Least Squares Policy Iteration (LSPI), which was introduced by Lagoudakis in [21] and [22]. This method can be used both model free and model based and uses a least squares approximation of the basis functions weights, to update the value approximation. Palunko [10], [16] and Shaker [15] show applications of this method to quadrotor control tasks.

At first the algorithm evaluates the value of the current policy using Equation 4-12.

$$Q(s,a) \approx \hat{Q}^{\pi}(s,a,w) = \sum_{i=1}^{k} \phi_i(s,a)w_i = W^T\Phi(s,a) \tag{4-12}$$

To update the matrix W, the squared error is minimized using Equation 4-13.

$$W = (\Phi^T(\Phi - \gamma P^{\pi}\Phi))^{-1}\Phi^T R \tag{4-13}$$

The transition matrix P, and reward vector R, are unknown in the model free case however. To fix this, an approximate form of Φ, $P^{\pi}\Phi$ and R is made using a set of samples $D = \{\{s_i, a_i, s'_i, r_i\}|i = 1, 2, ...L\}$. These samples might be acquired from the history of samples, or from training data of a human demonstrator.

Using this, the followings equation provide the approximations.

$$\widetilde{\Phi} = \begin{bmatrix} \phi(s_1, a_1)^T \\ \dots \\ \phi(s_L, a_L)^T \end{bmatrix} \tag{4-14}$$

$$\widetilde{P^{\pi}\Phi} = \begin{bmatrix} \phi(s'_1, \pi(s'_1))^T \\ \dots \\ \phi(s'_L, \pi(s'_L))^T \end{bmatrix} \tag{4-15}$$

$$\widetilde{R} = \begin{bmatrix} r_1 \\ \dots \\ r_L \end{bmatrix} \tag{4-16}$$

The policy then is updated, finding the action that maximizes the approximate Q-function.

$$\pi(s|w) = \arg\max_a W^T\Phi(s,a) \tag{4-17}$$

When LSPI is used online, there is the need for exploration. The used samples can be updated with the time history of experienced states and their respective rewards and follow up states. Palunko [10] uses LSPI online. The formulation used for the least squares is a little bit different, and the exploration is added in the algorithm. Algorithm 4 shows the working principle applied by Palunko. In this algorithm the matrices Λ, Γ and vector z (line 7) are introduced, they are discussed in detail in the book of Busoniu [11]. In this online variant the policy is updated every K_θ (line 15-18). When K_θ is 1, the weights are updated every time step. K_θ can be chosen bigger than 1, however, since the performance could even be better when K_θ is larger than 1 [10].

A big advantage of the use of LSPI for quadrotor control tasks, is that it does not need careful tuning of initial parameters. There is no learning rate and the algorithm does not

Algorithm 4 Online LSPI with ϵ-greedy exploration [10], [23]

```
1: Input: discount factor $\gamma$
2: BFs $\phi_1, \dots, \phi_n$
3: Policy improvement interval $K_\theta$
4: Exploration development $\{\epsilon_k\}_{k=0}^{\infty}$
5: $\beta_\Gamma$ a small constant
6: $l \leftarrow 0$
7: $\Gamma_0 \leftarrow \beta_\Gamma I_{nxn}, \Lambda_0 \leftarrow 0, z_0 \leftarrow 0$
8: Measure initial state $x_0$
9: for every time step $k = 1, 2, \dots$ do
10:     $u_k \leftarrow \begin{cases} \text{exploit: } \arg\max_a \phi^T(s,a) w_l \text{ with prob } 1-\epsilon_k \\ \text{explore: random action with prob } \epsilon_k \end{cases}$
11:     Apply $a_k$ and observe next state $s_{k+1}$ and reward $r_{k+1}$
12:     $\Gamma_{k+1} \leftarrow \Gamma_k + \phi(s_k, a_k)\phi^T(s_k, a_k)$
13:     $\Lambda_{k+1} \leftarrow \Lambda_k + \phi(s_k, a_k)\phi^T(s_{k+1}, \pi_l(s_{k+1}))$
14:     $z_{k+1} \leftarrow z_k + \phi(s_k, a_k) r_{k+1}$
15:     if $k = (l+1)K_\theta$ then
16:         $w_l \leftarrow$ solve $\frac{1}{k+1}\Gamma_{k+1} w_l = \gamma \frac{1}{k+1}\Lambda_{k+1} w_l + \frac{1}{k+1} z_{k+1}$
17:         $l \leftarrow l + 1$
18:     end if
19: end for
```

take gradient steps, it just minimizes the least squared error. This advantage comes at the price that a history of samples is used, so at the beginning of an episode, some samples should already be available [15]. These samples can be obtained in multiple ways, using human expert demonstration for this is a suitable option.

Approximate value iteration

Faust [3], [14] uses the approximate value iteration algorithm. As mentioned in Chapter 3, the difference between value iteration and policy iteration methods is that value iteration is off-policy, i.e., the policy is evaluated looking at the maximum next value, instead of using the current policy, as is done in policy iteration. In Algorithm 5, the AVI algorithm used by Faust in [3] and [14] is presented. The algorithm is offline and has the goal to update the basis function weights, to use this in a greedy policy afterwards. Within the learning part, it can be seen that in every iteration, directly after evaluating the optimal value for the current set of states, the weights of the basis functions are updated. This is done in order to converge to weights that best describe the optimal value function. Furthermore, it can be seen that this algorithm is model based; it uses a transition function.

The policy using this approximate value iteration scheme is a greedy policy π, which is used to generate a trajectory and to control the quadrotor. $\pi(s) = \arg\max_a (W^T \Phi(f(s,a)))$.

This offline method has a number of advantages according to Faust [3]:

1. The induced greedy policy is robust to some noise.

Algorithm 5 Least squares approximate value iteration (modified version of alg 3.1 from Busoniu [11])

```
Input: transition function $f(s,a)$, reward function $\rho$, discount factor $\gamma$, vector of basis functions $\Phi$ and samples $\{(s_{l_s})|l_s = 1,\dots,n_s\}$
Initialize parameter vector, e.g. $W_0 \leftarrow 0$
repeatat every iteration $l = 0,1,2,\dots$
    for $l_s = 1,\dots,n_s$ do
        $V_{l+1}(s_{l_s}) \leftarrow r(s) + \gamma \max_a [W_l^T \cdot \Phi(f(s_{l_s},a))]$
    end for
    $\theta_{l+1} \leftarrow \arg\min_W \sum_{l_s=1}^{n_s} \left(V_{l+1}(s_{l_s}) - W^T \cdot \Phi(s_{l_s})\right)^2$
until maximum number of iterations is performed
```

2. The policy is agnostic to the simulator used.
3. The algorithm can learn on a subset of the domain.

These advantages make the AVI algorithm suitable for an offline use. An online use has not been found in literature.

Continuous Action Fitted Value Iteration

Whereas the previously described method discretizes the action space, the method proposed by Faust [24] and Figueroa [12] can be used with a continuous action space. The method is similar to the approximate value iteration approach and is called Continuous Action Fitted Value Iteration (CAFVI). In order to be able to use CAFVI, the system should be control-affine i.e., equation 4-18 should hold.

$$D:\ s_{k+1} = f(s_k) + g(s_k)a_k \tag{4-18}$$

Where f and g are some function of the state. For this approach to work, the basis functions should be quadratic functions. In this sense the value function can be viewed as a Lyapunov function. The algorithm used in this approach is presented in Algorithm 6. In line 8 and 9 it can be seen that this approach actually uses system dynamics samples to estimate the action-value function. This samples can be obtained using a simulator, but more interesting also by means of a human demonstrator. The update of the weights of the basis functions is done by minimizing the least square error between the current estimate of the value with the current weights of the basis functions, and the estimated value found by using the value of the next state for each of the samples (line 15-16).

To find the action yielding the maximum of Q, equation 4-19 is used, where $q_i = [Q^p_{s,1}(a_{i1}); Q^p_{s,2}(a_{i2})\ Q^p_{s,3}(a_{i3})]$ are three samples of $Q^p_{s,i}(a)$, obtained at points $[a_{i1}\ a_{i2}\ a_{i3}]$ running through point p.

$$\hat{a}_i = \min(\max(\hat{a}^*_i, a^l_i), a^u_i) \tag{4-19}$$

$$\hat{a}^*_i = \frac{q_i^T \cdot ([a_{i2}^2\ a_{i3}^2\ a_{i1}^2] - [a_{i3}^2\ a_{i1}^2\ a_{i2}^2])^T}{2q_i^T \cdot ([a_{i2}\ [a_{i3}\ [a_{i1}] - [a_{i3}\ [a_{i1}\ [a_{i2}])^T} \tag{4-20}$$

On the interval $a_i^l \leq a \leq a_i^u$, where superscripts u and l denote the upper and lower value respectively.

Algorithm 6 Continuous Action Fitted Value Iteration [24]

```
1: Input: $S, A, \gamma$
2: Input: basis function vector $\Phi$
3: Output: $W$
4: $W_0, W_1 \leftarrow zerovector$
5: $l \leftarrow 1$
6: while $l \leq maxiterations$ AND$||W_l - W_{l-1}|| \geq \epsilon$ do
7:     for $l_d = 1, \dots, n_d$ do with $n_d$ is the number of samples
8:         sample state $s_{l_d}$ and observe its reward $r_{l_d}$
9:         $\{s_{l_d}, a_{ij}, s'_{ij} | i = 1, \dots, d_a, j = 1, 2, 3\}$ (obtain system dynamics samples)
10:        for all i and j: $q_{ij} \leftarrow W_l^T \Phi(s'_{ij})$ (estimate action-value function)
11:        $\hat{a} \leftarrow$ calculated using equation 4-19
12:        Apply $\hat{a}$ and find the full tuple $\{a_{l_d}, \hat{a}, s'_{l_d}, \rho_{l_d}\}$
13:        $v_{l_d} = r_{l_d} + \gamma W_l^T \Phi(s'_{l_d})$ (state-value function new estimate)
14:    end for
15:    $W_{l+1} \leftarrow \arg\min_W \sum_{l_d=1}^{n_d} (v_{l_d} - W^T \Phi(s_{l_d}))^2$
16:    $l \leftarrow l + 1$
17: end while
```

The advantage of CAFVI is that it allows for continuous action, which results in smoother control. A downside of this approach is that there are more restrictions on the choice of the reward function and the basis functions, in order to assure Lyapunov stability. “But, basis functions must fit reasonably well into the true objective function determined by the system dynamics and the reward, otherwise CAFVI diverges.” [24]. In the applications of CAFVI by Figueroa [12] and Faust [24], no actual use is made of human demonstrations, instead a simple simulation model is used. As mentioned, CAFVI does allow for the use of human demonstrations.

Learning Automata

A slightly different algorithm that can be model free, is Learning Automata, as used by Santos [25]. This approach works for a finite action set and is by Santos used to tune the PID gains for a quadrotor, in order to allow for stable hover and simple trajectory following. The principle of learning automata is to give each action a probability of being selected. Following from the received reward, the probability of the action is adjusted: it will be increased when the reward is high, but decreased when the resulting reward is low. In this way, the probabilities of selecting actions is iteratively updated, ultimately, for each given state to make the probability of one of the actions converge to one.

When this method is implemented online, it could be considered as a purely model free reinforcement learning algorithm. For a quadcopter hover task this way of learning is not suited, since it will crash a lot before it has learned the proper policy. As a result, Santos [25] uses this method offline in simulation. When the policy has converged in simulation, it is used in an actual real life experiment. This policy yields acceptable results for the hover task, but it would not be able to cope with changing conditions.

Dynamic programming approaches

Some of the aforementioned methods are methods that can be applied without a transition model of the system, though they can also be applied with a transition model. There is a class of algorithms that does not function without a proper model of the system, these are approaches using dynamic programming. In the work of Abbeel [8] and Palunko [19], this approach is used in order to find a good policy.

Dynamic programming works using the principle of optimality. When the full MDP is known, similar to some of the model free methods, the policy will be iteratively improved towards the optimum. This can be for example done using policy iteration, which is shown in Algorithm 7. By this, the Bellman equation is solved iteratively to find the optimal policy $\pi^*(s)$. For a more elaborate explanation on dynamic programming approaches, the book of Sutton and Barto [4] is a good introduction, and the book of Busoniu [11] is a good extension on this for large and continuous state spaces.

Algorithm 7 Dynamic programming policy iteration

```
Inputs:
Initialize $Q(s,a)$ and $\pi(s)$
while $\pi(s)$ is still changing do
    while $Q(s,a)$ is still changing do
        $Q(s,a) \leftarrow \rho(s,a) + \gamma Q^{\pi}(s',a')$
    end while
    $Q^{\pi}(s,a) = Q(s,a)$
    $\pi(s) \leftarrow \arg\max_a Q^{\pi}(s,a)$
end while
$Q(s,a) = Q^*(s,a)$, $\forall s$ and $\forall a$
$\pi(s) = \pi^*(s)$, $\forall s$
```

4-2 Model based applications

Having looked at the different components of reinforcement learning methods that are used for quadrotor control, in this section some model based examples of algorithms will be studied, to see how these components can be used together. In section 4-2-1 some algorithms that use, or could use, human expert demonstrations within their algorithm are discussed. In section 4-2-2 some examples that do not use human expert demonstrations are presented.

4-2-1 Model based algorithms that (can) use human expert demonstration

Policy Iteration using a least squares transition model deducted from demonstrations

The first model based algorithm case study is found in a paper of Waslander [20], where an Integral Sliding Mode controller is compared to a reinforcement learning controller. The task is to control the altitude of a quadrotor. The set-up of the reinforcement learning algorithm consists of a regulative reward function, which penalizes deviations from a reference altitude and velocity. In this algorithm no explicit value function is used. Since this is an offline model based algorithm, complete episodes are evaluated in simulation, and so the total return after an episode is used as a measure of comparison between policies. The transition model used for this algorithm is a locally weighted linear regression, on either a human expert demonstration set, or another training set. The policy basis is a random one, with the use of a polynomial describing the policy as function of the states. The weights within the polynomial are adjusted to find better policies.

The complete reinforcement learning algorithm is presented in Algorithm 8.

Algorithm 8 Model Based Reinforcement Learning [20]

```
 1: Generate set $S_0$ of random initial states
 2: Generate set $T$ of random reference trajectories
 3: Initialize $w$ with reasonable initial guess
 4: $R_{best} = -\infty$, $w_{best} = w$
 5: repeat
 6:     $R_{total} = 0$
 7:     for $s_0 \in S_0, k \in T$ do
 8:         $S(0) = s_0$
 9:         for $k = \{0, 1, 2, \ldots, K-1\}$ do
10:             $a_k = \pi(s_k, w)$
11:             $s_{k+1} = f(s_k, a_k) + v$
12:             $R_{total} = R_{total} + \rho(s_k, a_k)$
13:         end for
14:     end for
15:     if $R_{total} > R_{best}$ then
16:         $R_{best} = R_{total}, w_{best} = w$
17:     end if
18:     Add Gaussian random vector to $w_{best}$, store as $w$
19: until $w_{best}$ converges
```

From the algorithm it can be seen that this approach has, similar to genetic algorithms, a solution around which randomly a new solution is sampled. This approach makes the method, in terms of speed of convergence, sensitive to the initialization of the policy function weights. The advantage of this approach is that it can escape local maxima and find the global maximum. Waslander concludes that using this approach “The reinforcement learning control law is susceptible to system disturbances for which it is not trained.”

The black box system identification method used, Locally Weighted Linear Regression (LWLR), shows to be a good extension on an ordinary least squares solution. In the domain of states in which a quadrotor operates, a lot of variety is present. The local weighting in the regression makes it possible to fit a good model in the presence of local effects. An analysis of the model in, for example, ground effect showed a good representation by the model.

Concluding, the big advantage in this paper is the use of human expert demonstrations ,in combination with the LWLR model, which shows promising results. The learning algorithm itself is used offline, so is less suitable considering the demand of performing well in changing conditions. An online extension of the algorithm could be a solution to this. It uses an initial policy found offline using the method described before, and then online continues learning, updating the policy weights after each episode.

Using expert pilot demonstrations in helicopter acrobatics

The only method described in this literature survey which is not applied to a quadrotor, is the work of Abbeel [8]. There are many differences between helicopter and quadrotor dynamics, however, they are both complex. The reason this helicopter application is studied, is that it distinctly uses human expert demonstrations for complex maneuvers.

The tasks pursued in this paper are challenging; acrobatic maneuvers such as front flips and side rolls. The algorithm consists of the following components. The reward function is regulative. It contains the errors of position, velocity and rotations over all different axes as well as the control inputs. Similar to the effort of Waslander [20], the algorithm is used offline. As such it does not use an approximation of the value, but uses the cumulative reward over an episode instead, to evaluate a policy. The transition function is derived using a so called grey-box [18] system identification method. This means that a simple model structure is assumed according to the dynamics of the helicopter. The parameters within this model structure are found using linear regression, with as training input the data of a set of human expert demonstrations. The policy is found using differential dynamic programming which has a policy iteration approach.

Evaluating the work of Abbeel, first of all it should be noted that the final implementation of the algorithm has lead to successful autonomous execution of some advanced aerobatic maneuvers. The grey-box system identifications shows potential, since it includes some prior knowledge to the system, which in combination with the expert demonstrations yields good results. It was found that, since these maneuvers are rather complex, and the algorithm does not learn anymore while flying, some of the rewards had to be hand-tweaked, to end up with satisfactory results [8]. This definitely is a downside of the chosen, model based approach.

4-2-2 Model based algorithms that do not use human expert demonstration

Offline approximate value iteration using a simple white-box model

In the paper of Faust [3], the task described is trajectory generation for a quadrotor with a slung load, in order to reach a target location with minimal swing of the load. In this algorithm, the reward function is mission specific. It contains the distance to the goal, i.e., the closer to the goal, the lower the penalty. Furthermore, it contains the swing angle, so that the penalty increases with an increasing swing angle, in order to minimize the swing of the load.

The basis functions are mission specific. The only basis functions used are the distance towards the goal, the velocity, the swing angle and the swing angular velocity. The advantage of mission specific basis functions becomes clear; only a few basis functions are needed, since they are directed towards the mission.

This algorithm uses a simple transition function. A transition function is used, since only the state value is estimated with the basis functions, and not the state-action value. Hence a way of calculating the next state is needed, to determine what the best action is. In this case, a simple white-box model is used (explained in Equation 4-6).

The policy is learned offline using a simulator. The learning is done using approximate value iteration, which provides an approximate value function. The policy employed on the physical task is fully greedy towards the approximate value function: $\pi(s) = \arg\max_a (W_T \cdot \Phi(f(s,a))$ where $f(s,a)$ is the aforementioned transition function.

The results of simulations and experiments show that the learned agent reaches the goal with an acceptable performance. Whereas the general trend in the trajectory in experiment and simulation is similar, the experiment shows much bigger oscillations. This can be explained by the fact that the simulator used is not a perfect model of reality, and hence some effects are not properly assessed by the simulator. In order to cope with this difference in simulator results and experiment results, noise was added to the simulator dynamics. This addition did bring the results of the simulator closer to those in the experiment, however, still significant differences are present. This result clearly shows the limitations of model based methods.

Learning automata in a model based way

Santos [25] uses the principle of learning automata to control a quadrotor in hover. The actions are defined differently in this work compared to the aforementioned efforts. Where in Faust [3] and Abbeel [8] the actions are actual control actions (forward and backwards acceleration for example), Santos approaches this a little bit differently; the action space is defined as an equidistant distribution over the proportional, integral and derivative gains of a PID controller on the orientation angles and position of the quadrotor. The reward following from an action, therefore, is composed using the average attitude over four seconds with a particular combination of gains in terms of steady state error, overshoot and cumulative error, which is definitely regulative.

The approach of Santos which tunes PID gains in simulation by evaluating the performance of a set of gains over a window of time, is different from other approaches discussed in this literature study. It shows that reinforcement learning approaches can also be used in combination with conventional PID control. It should be noted though, that this would only be possible offline in this case, since a quadrotor which starts flying with randomly initialized PID gains, will have a small chance of hovering well enough not to crash at the first iteration.

4-3 Model free applications

The following sections will describe some case studies of model free algorithms used in quadrotor control. In section 4-3-1 some algorithms that use, or could use, human expert demonstrations within their algorithm are discussed. In section 4-3-2 some examples that do not use human expert demonstrations are presented.

4-3-1 Model free algorithms that (can) use human expert demonstration

Shaker: least squares policy iteration using human demonstrations

The first model free approach studied is the effort of Shaker [15], which describes the use of a Least Squares Policy Iteration (LSPI), with continuous states and actions, to learn a quadrotor in simulation to land based on computer vision.

The reward function is regulative, which rewards when the goal is reached, but also gives a reward which gradually changes when getting closer to the goal. The Q-function is approximated using generic basis functions. Two different sets of basis functions are used, Radial Basis Functions and Polynomial Basis Functions, which both show good results. As this is a model free algorithm, there is no transition function used. The policy iteration algorithm is greedy towards the Q-function, which is approximated using the basis functions. The Q-function is updated by changing the basis function weights. In most examples in literatur,e LSPI considers discrete actions, in this algorithm a continuous action space is implemented. This gives the advantage of allowing for smoother trajectories.

A downside of the work of Shaker is that this approach has not been tested on a real MAV in experiment, but only in simulation. In simulation the policy converged, fastest with the use of radial basis functions, and showed promising results. Interesting is the fact that the algorithm uses samples to learn a good policy with the LSPI algorithm. Shaker does not use human demonstrations to collect these samples, but a simulator. It would, however, be possible to obtain the samples from human demonstrations.

Continuous action fitted value iteration with potential use of human demo's

Faust [24] uses the Continuous Action Fitted Value Iteration (CAFVI) approach for learning two tasks: A cargo delivery task with a cable suspended load, and a rendezvous task of a quadrotor slung load with a ground robot vehicle. In this section the cargo delivery task will be analyzed.

The reward function used in this algorithm is a regulative one; it rewards reaching the end goal, but also penalizes the distance to the end goal, and the sling angle. The basis functions are the same as in the previously described work by Faust [3]; they use the distance to goal, velocity, load displacement and load angular velocity, which is a mission-specific set of basis functions.

This is a model free method, and hence has no need for a transition function. The CAFVI algorithm does need samples however. They can be obtained using human expert demonstrations or by a simulator. In this case the state is randomly sampled using a simple simulator, which is the same as in Equation 4-6.

Three different way of defining a policy are used in this approach, which are different from the usual ($\epsilon-$) greedy policies. First is the Manhattan policy, which determines an action in a serial way; it finds the maximum of Q on one axis in the state space, and from that point on finds maxima on the subsequent axes. The second method is called the convex sum policy, which is a convex combination of the maximums found on each axis independently. Finally, there is the axial sum policy, which is a maximum of the policy approximation obtained using the convex sum policy and non convex axial combinations. The policies do not have a distinct way of exploring such as ϵ-greedy policies, but do so because the policies are sub optimal, and hence explore a bit around the optimal policy. The update of the policy is done using the value function update, which is done using the CAFVI algorithm as described in Algorithm 6.

To evaluate the performance of the different policies within the CAFVI algorithm, they are compared to each other and to a discrete AVI method similar to previous work of Faust [3]. It is shown that the computation time for the CAFVI approach is independent of the input dimensionality, in contrast to the discrete AVI method. This shows a big advantage for high dimensional action spaces.

The results on the cargo delivery show that the convex sum policy performs best on a number of criteria. The computational time for this policy is lower than for the others, and lower than for the discrete AVI method. Moreover, it performs best in terms of minimal load swing and in energy needed for control. On the other hand, the axial sum policy performs better in terms of time to goal and delivery precision. In all these criteria mentioned, the CAFVI method reaches better performance than the discrete AVI approach.

A downside of this method, as mentioned, is that there are criteria on the reward and basis function to ensure convergence. Furthermore, the method is a little less transparent than for example the AVI algorithm, which has a more straightforward policy and approach, in which it is easier to analyze what happens within the algorithm. An interesting feature of the CAFVI algorithm, is the possibility of using human demonstrations as samples.

4-3-2 Model free algorithms that do not use human expert demonstration

Online least squares policy iteration

In [10], Palunko tries to minimize the tracking error of a slung load, suspended from a quadrotor, while also minimizing the swing angle of the cable. This means that the algorithm has to find a sequence of control actions for the quadrotor, that ensures that the load follows a prescribed trajectory.

For this task, the reward function is a continuous one specific to the mission. It penalizes the error of the load with respect to the reference trajectory, the load swing angle and the control action, by that minimizing the control effort. The basis functions in this algorithm are general, they are chebyshev polynomials of the first kind, which include state and action, and as such constitute a state-action value function. The policy is ϵ-greedy towards the Q-function. The update of the policy is done every seven iterations, using the approach described in Algorithm 4.

The performance on the prescribed task is tested both in simulation and in experiment. In the simulation the performance of the algorithm is good. The reference trajectory is followed well and the slung angle stays well within 7 degrees, oscillating a little bit around the equilibrium position. The same goes for the experimental results, which show a good performance of the algorithm, albeit with a little higher deviations from the reference than in simulation. An interesting test done in simulation for this algorithm is a change in the conditions, to see how the algorithm responds to changes in the environment. To this end the length of the cable was suddenly changed in the simulation. The results show that in a short time the new situation is anticipated for. This result shows the power of a model free and online algorithm. It anticipates on changing conditions, because it is not using a model of the environment and keeps updating its policy while interacting with the environment.

4-4 Performance metrics for comparison of methods

In literature, different criteria are used to evaluate the relative performance of various algorithms. In general, two main criteria are of interest:

1. The time/number of iterations it takes to learn a good policy. This criterion tries to answer the question: How fast can the algorithm learn?
2. The performance of the final policy on the task at hand. This criterion tries to answer the question: How well does the final policy perform the task?

In literature on quadrotor reinforcement learning control, mostly qualitative comparisons are made between different algorithms. The majority of papers only looks at the time history of different state parameters, to evaluate performance at a certain task, e.g., Waslander [20] shows the reference altitude the quadrotor should follow, and the actual altitude the quadrotor is flying as a function of time. The conclusion drawn from the plots is as follows: "Comparison of the step response for Integral Sliding Mode (ISM) and RL control reveals both stable performance and similar response times, although the transient dynamics of the ISM control are more pronounced." [20]. This is a qualitative way of evaluating performance and compare different methods. To specifically compare different methods with each other, there is a need for methods that are quantitative and clearly defined. A few papers use a quantitative way of evaluating performance of algorithms, considering either the time to learn, or the performance of the final policy found by the algorithm.

4-4-1 Time to learn

Faust [3] uses the offline CAFVI algorithm with different policy structures, and compares them to discrete AVI. Since these policies are all offline, the number of episodes needed to converge is a measure for the time to learn a policy. Since the computation time per iteration can differ between methods, the actual time to learn might be a better criterion to compare algorithms. The absolute time to learn is dependent on the computer on which the computations are carried out. This means that comparing the time to learn does not give a reproducible result. To this end Shaker [15] and Kaelbling [26] do not use the time to learn, but the computational complexity of the operations needed. "A standardized measurement of the computational time complexity of an algorithm is the number of elementary computer operations to solve a problem, in the worst case." This is a more cumbersome approach, but does provide reproducible comparisons.

In online algorithms, the time to learn is more critical than in offline algorithms. In offline algorithm the time to learn is mostly a matter of needed computational resources. For online algorithms, the time to converge to a satisfactory result is critical, since the performance of the task is most likely low, when a good policy has not been found yet.

4-4-2 Performance

The performance of an algorithm is apart from the time to learn, also determined by the quality of the found policy. In the paper of Faust [3], the performance of the different algorithms is compared for a cargo delivery task. The algorithm runs offline, so the policy found after learning is evaluated. Several different criteria are studied: The load displacement, the time to reach the goal, and the energy needed to control the quadrotor. This is a method of separately evaluating the goals for the task at hand. For each of the goals a winner is found. This approach is a common approach for comparing different algorithms [24].

A more general approach to compare the performance, especially for online algorithms, is measuring the accumulated return for a task. This approach is also used in an online algorithm by Faust [24], and uses the initial goals set within the algorithm: the rewards. This method allows to compare different algorithms, using the same reward function in a general sense.

Concluding, the performance of the policy found by the algorithm is the most important criterion comparing different offline algorithms, since this is the policy that will be used in actual flight. For online algorithms this same criterion is also important. However, for online algorithms the accumulated return is also an important criterion, as this is a measure of how good or bad the policy performed, while it was still developing.

4-5 Conclusions on the literature review

A literature study is conducted to identify different reinforcement learning approaches which are used in quadrotor control. Within these methods, the use of human expert demonstrations is studied, as well as the way in which performance is measured and compared between different algorithms.

It is found that several different reinforcement learning techniques have been used in quadrotor control. A distinction can be made between model based and model free algorithms. Furthermore, a distinction is made between algorithms that are offline, and algorithms that run online, and as such can adapt to changing conditions.

In quadrotor control, there is as far as this study reached only one mention found of an algorithm that uses human expert demonstrations within a reinforcement learning algorithm. Waslander [20] describes an algorithm that runs offline, and makes use of flight data of a pilot flying the quadrotor. The algorithm used is offline and model based, and thus not capable of adapting to changing conditions. Abbeel [8] describes the explicit use of human expert demonstrations within reinforcement learning as well. The application of this is a helicopter however. Furthermore, it should be noted that similar to Waslander, the algorithm used by Abbeel is an offline, model based method, and as such is not capable of adaptation to changing conditions.

In some algorithms found, there would be the possibility to use expert demonstration, but as far as the corresponding papers describe, a simulator was used to generate input-output data, and no human demonstrations. Faust [24] makes use of continuous action fitted value iteration, which uses input-output samples to calculate an update of the policy. The explicit use of human demonstration is not mentioned in this paper. Furthermore, the method runs offline, which makes it not able to adapt its behavior. Shaker [15] on the other hand, uses least squares policy iteration, in which the basis function weights are calculated online, by using an initial set of samples, without using a model of the dynamics of the system. Human expert demonstrations can help to make the policy start with a good guess of a policy. Concluding, a good implementation of human expert demonstration within a reinforcement learning algorithm for quadrotor control, which is able to adapt to changing conditions, is not yet available.

In the literature study the different components of a reinforcement learning algorithm, as used in quadrotor control, are discussed. It can be concluded that when the wish to have an algorithm that is generally applicable (different tasks, extensible to different vehicles), it helps to use an open reward function, and general basis functions. For a policy base and improvement, LSPI and AVI are found to be good options. They are converging rather fast towards a good policy, and allow for the use of human demonstrations.

Finally, it is found that up until this moment few different performance metrics are used in literature to quantitatively assess the performance of a reinforcement learning algorithm on a particular task. A measure which is good for comparing different methods is the cumulative reward, as long as the same reward function is used. In addition to this a measure of “time to learn” is useful, to value methods that converge faster.

Chapter 5

Preliminary Simulation

In the preliminary simulation the feasibility is evaluated of the approach proposed in the literature study, using a simplified version of the quadrotor slung load problem. The simulation is two dimensional, with only lateral movement of the quadrotor x, and swing angle of the cable θ. The task in this simulation is to drop the slung load at the target as quickly as possible. A snapshot of the simulation can be seen in Figure 5-1. Since the quadrotor cannot move up or down, it has to swing the load up to get it in the target basket.

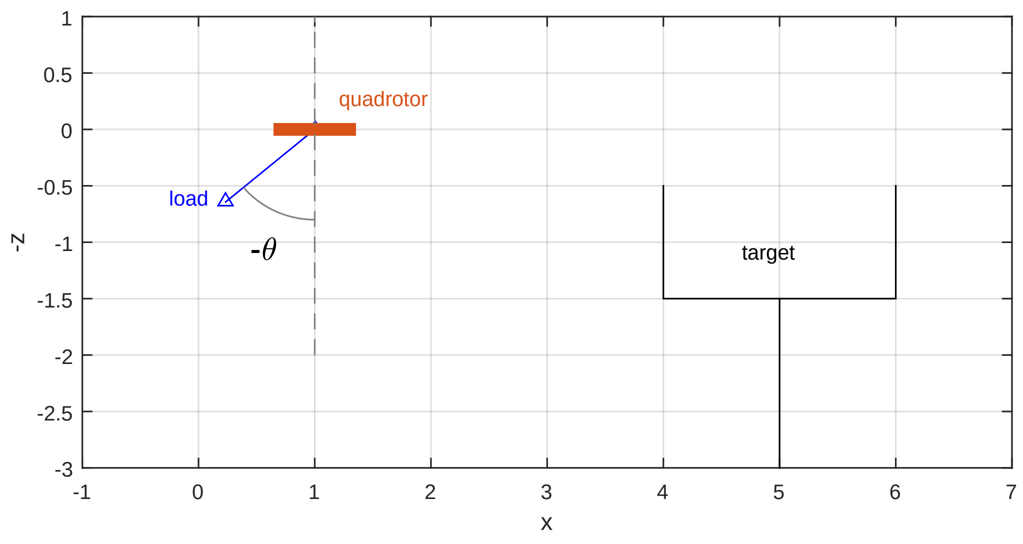


Figure 5-1: Snapshot of the preliminary 2D quadrotor simulation

In section 5-1 the simulation setup will be described after which the implemented algorithms are discussed in section 5-2. Section 5-3 shows the results for the implemented algorithms on this 2D problem. Finally, in section 5-4, the values found by the algorithm are studied, as an extra check on the reliability of the results, and to see if some behavior can be generalized to be used for constructing basis functions.

5-1 Simulation setup

For this 2D preliminary simulation, the position of the quadrotor is defined in an inertial frame, with x pointing right and z pointing downwards. The location of the pendulum is described in a radial reference frame, centered in the c.o.m. of the quadrotor. When the pendulum is hanging down, $\theta = 0$ and swinging to the right corresponds to positive θ.

First, the motion of the quadrotor is considered. The influence of the slung load on the quadrotor is neglected. Since the quadrotor can only move along the x-axis, its position is fully described by its x position x_q.

The control input for the system is the acceleration of the quadrotor. It is assumed that this is the only net acceleration on the quadrotor.

$$\ddot{x}_q = a \tag{5-1}$$

The state is propagated using a simple Euler integration scheme with time step T_s.

$$\dot{x}_q = \dot{x}_{q0} + \ddot{x}_q \cdot T_s \tag{5-2}$$
$$x_q = x_{q0} + \dot{x}_q \cdot T_s \tag{5-3}$$

The angular acceleration is calculated as follows: [3]

$$\ddot{\theta} = \begin{bmatrix} -\cos\theta & \frac{\sin\theta}{L} \end{bmatrix} \cdot \begin{bmatrix} \ddot{x}_q \\ -g \end{bmatrix} \tag{5-4}$$

The angle is propagated using an Euler integration scheme as well.

$$\dot{\theta} = \dot{\theta}_0 + \ddot{\theta} \cdot T_s \tag{5-5}$$
$$\theta = \theta_0 + \dot{\theta} \cdot T_s \tag{5-6}$$

In this simple simulation, a couple of assumptions are made concerning the slung load. The cable is assumed to be inelastic and mass less. Furthermore, it is assumed that the cable is suspended from the c.o.m. of the quadrotor. The system is assumed to be undamped, so aerodynamic forces are neglected, as well as friction of the cable in the suspension point. Furthermore, it should be noted that the cable is actually modeled as a rod, since the simulation does not allow the cable to become slack.

5-1-1 State and action

The state of the quadrotor with slung load is represented by its position, velocity, the slung angle of the load and the angular rate of the slung load. To keep the preliminary simulation of the reinforcement learning algorithms simple, the state variables are discretized and capped to certain minimum and maximum values, which are presented in Table 5-1.

Table 5-1: Discretization of the states of the system

	minimum	maximum	spacing	number of entries
x_q [m]	-2	8	0.5	21
$\dot{x}_q$ [m/s]	-5	5	0.5	21
θ [deg]	-110	110	10	23
$\dot{\theta}$ [deg/s]	-540	540	20	55

5-1-2 Definition of goal state

When the load is dropped into the target basket, the simulation is stopped. To reach the goal, the following conditions should be fulfilled at the same time:

- The x position of the load is within $x_{goal} \pm x_{margin}$ where x_{goal} is the basket center, and x_{margin} defines its width.
- The z position of the load must higher than z_{goal} but lower than $z_{goal} + z_{margin}$
- The load must be moving downwards ($z_l \geq 0$)

In short, the load must fall in the basket from the top.

5-1-3 Used dimensions and constants

In Table 5-2 the constants used in this simulations are presented. The reward structure used in this problem is kept as simple as possible. A reward is given when the goal is reached. As the goal is to drop the load in the basket as quickly as possible, the time to reach the goal is included in the reward. Since a sling angle of higher than 90 deg is not realistic, because this would cause the cable to tangle up with the propellers, a too high sling angle is penalized in the reward structure. The reward structure is formalized in 5-7.

Table 5-2: Used dimensions and constants

Variable	Value	Unit	Description
g	9.81	m/s^2	Gravitational acceleration
L	1	m	Cable length
T_s	0.2	s	Time step
T_{end}	5	s	Simulation time
x_{start}	0	m	Starting position
z_{start}	0	m	Starting position
x_{goal}	5	m	Goal position
z_{goal}	0.5	m	Goal position
x_{margin}	1	m	Goal margin
z_{margin}	0.3	m	Goal margin

$$r = \begin{cases} -t_{goal}^2 \text{ if goal reached} \\ -2T_{end}^2 \text{ if goal not reached} \\ -10 \text{ if } \theta > 80 \text{ deg} \end{cases} \tag{5-7}$$

5-2 Implemented algorithms

As mentioned before, two algorithms are implemented within this preliminary simulation. As a benchmark a simple Q-learning algorithm is implemented. The second algorithm is a reinforcement learning algorithm that uses human demonstrations.

5-2-1 Q-learning implementation

The benchmark reinforcement learning algorithm used in this research is discrete Q-learning with ϵ-greedy exploration. Algorithm 9 shows the working principle of Q-learning. As can be seen in lines 13-16, the Q-values are updated only at the end of an episode. This is done because at that moment, the exact return is known for the visited state action-combinations.

Algorithm 9 Q-learning with ϵ - greedy exploration and MC updates

```
 1: Input: number of iterations $K$, simulation time $T_{end}$, timestep $T_s$
 2: exploration and learning rate schedule $\{\epsilon_k\}_{k=0}^{\infty}$
 3: Action space $A$
 4: Initialize state $s_0$
 5: Initialize $Q(s,a)$ with zeros
 6: for $k = 1 : K$ do
 7:     $s = s_0$
 8:     for $j = 1 : \frac{T_{end}}{T_s}$ do
 9:         $a_k = \begin{cases} a \in \arg\max_{\bar{a}} Q_k(s_k, \bar{a}), \text{ with probability } 1 - \epsilon_k \\ \text{uniformly random action in } A, \text{ with probability } \epsilon_k \end{cases}$
10:         apply $a_k$, measure next state $s_{k+1}$ and reward $r_{k+1}$
11:         Break if goal is reached
12:     end for
13:     for $j = 1 : j_{max}$ do
14:         $R_j = \sum_{n=j}^{j_{max}} r_n$
15:         $Q_{k+1}(s_j, a_j) = Q_k(s_j, a_j) + \alpha_k[R_j - Q_k(s_j, a_j)]$
16:     end for
17: end for
```

5-2-2 Q learning with human demonstration implementation

In LSPI for quadrotor control which is used by Shaker [15], the state is continuous, and estimated using basis functions. In this simple simulation a discrete state space is used, to allow for quick results and comparison. The problem, however, is that the proposed method of using human demonstrations in RL, is done with least squares estimation of the basis functions weights. By first using samples of human demonstrations, a good initial guess of these weights is made, and hence the algorithms performance is improved. Since in the discrete case, basis functions are not used, a slightly different approach is taken, still following the contours of the continuous state LSPI.

LSPI is mimicked by using the human demonstrations for initial guessing of the state-action values Q, see Algorithm 10. A number of human demonstrations is used, for which, using Monte Carlo update, the average value of all the visited state-action pairs is determined (line 6-11). Then instead of initializing the Q-matrix random, or with zeros, all the visited state-action values are formed by the values found using the human demonstrations (line 13). As such the initialization of the value estimation is improved using the human demonstrations, as is the case in the LSPI algorithm proposed by Shaker [15].

Algorithm 10 Q-learning with ϵ - greedy exploration and MC updates - Use preknowledge

```
1: Input: number of iterations $K$, simulation time $T_{end}$, timestep $T_s$
2: exploration and learning rate schedule $\{\epsilon_k\}_{k=0}^{\infty}$
3: Initialize state $s_0$
4: Initialize $Q(s,a)$ with zeros
5: *Here follows the initialization using expert demo's of the Q table*
6: for $n = 1 : N_{samples}$ do
7:     for $j = 1 : j_{max}$ do
8:         $R_j = \sum_{n=j}^{j_{max}} r_n$
9:         $Q_{n+1}(s_j,a_j) = Q_n(s_j,a_j) + \frac{1}{\#Visits(s_j,a_j)}[R_j - Q_n(s_j,a_j)]$
10:    end for
11: end for
12: *From here on Q-learning is applied*
13: $Q_0 = Q_{N_{samples}}$
14: for $k = 1 : K$ do
15:    $s = s_0$
16:    for $j = 1 : \frac{T_{end}}{T_s}$ do
17:        $a_k = \begin{cases} a \in \arg\max_{\bar{a}} Q_k(s_k,\bar{a}), \text{ with probability } 1-\epsilon_k \\ \text{uniformly random action in } A, \text{ with probability } \epsilon_k \end{cases}$
18:        apply $a_k$, measure next state $s_{k+1}$ and reward $r_{k+1}$
19:        Break if goal is reached
20:    end for
21:    for $j = 1 : j_{max}$ do
22:        $R_j = \sum_{n=j}^{j_{max}} r_n$
23:        $Q_{k+1}(s_j,a_j) = Q_k(s_j,a_j) + \alpha_k[R_j - Q_k(s_j,a_j)]$
24:    end for
25: end for
```

5-3 Results

5-3-1 Description of the human demonstration sets

In this preliminary simulation four sets of human demonstrations are used. These are generated using the simulation real-time with a joystick as input. A description of each set is given in Table 5-3. Each set consists of five demonstrations by the human pilot.

Table 5-3: Description of the demonstration sets

Set number	Description
1	First set with the demonstrator trying to perform best.
2	Second set with the demonstrator trying to perform best.
3	Set in which the demonstrator starts steering to the left, which is not optimal, and after this tries to perform best.
4	Set in which the demonstrator deliberately shows poor performance.

The applied inputs for the different demonstration sets are shown in Figure 5-2. Here it can be seen that especially the first set has low variation between the different trials. In demonstration set number 4, a lot more variation can be seen. Furthermore, it can be seen in set 3, that each trial starts with a movement to the left (a negative action) before moving the quadrotor to the right. As a measure for the type and quality of the sets of human demonstrations, in Figure 5-3, a boxplot is presented to show the mean return and variance within the set for the used demonstration sets. Especially the fourth set stands out, which is logical because here a lot of variation was used deliberately by the human pilot, and hence the scores were rather different.

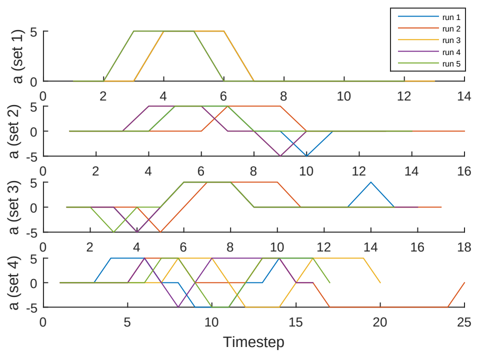


Figure 5-2: Input over time for the four different demo sets

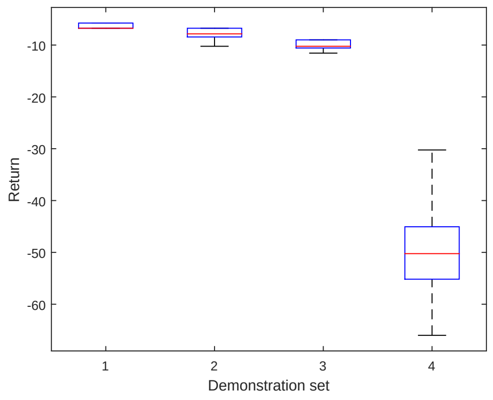


Figure 5-3: Boxplot of the return for the four different demo sets

5-3-2 Influence of initialization parameters

A couple of parameters used in both the Q-learning algorithm and the Q-learning algorithm with human demonstration can be tuned. The learning rate α, exploration rate ϵ, the number of iterations and the initialization of the Q-values have an important impact on the outcome of the algorithm. All these parameters are varied to find a good combination of them for satisfactory results. An analysis of these variations is provided in Appendix A. In short, it was found that higher exploration rates result in a faster convergence to an optimal policy. However, this comes at the cost of a lower average return, since a large portion of the actions is random, and hence often yields low returns. A good value for the exploration rate was found to be 1 %. The learning rate should be chosen not too high and not too low, a value of 0.01 gives good results. In general it was seen that 2000 iterations is enough to converge to a satisfactory policy, however, for lower learning rates, more iterations can be needed.

The initialization of the Q-values has an interesting influence. In pure Q-learning, it would make sense to initialize the Q-values with zeros. Since all actual trials result in a negative return, this approach will stimulate optimism in the face of uncertainty i.e., all states that have not been visited yet, have higher value than the states that are visited already. In the long run, for Q-learning, this results in finding an optimal policy. Putting the initial Q-values lower, means that the algorithm converges faster to a satisfactory policy, but not to the optimal one.

For the Q-learning combined with the use of human demonstrations, however, initializing the Q-values at zero does not make sense. It would mean that all states visited in the demos, which have gotten a value of lower than 0, will be skipped. This means that the contrary happens to what should happen using the human demonstrations. As such, a value of -50 as initialization of the Q-values is chosen.

5-3-3 Difference between Q-learning and Q-learning using human demonstrations

Using the aforementioned initialization, the Q-learning algorithm is ran for 2000 iterations, as well as the Q-learning with the use of the four different human demonstration sets. Figure 5-4 shows the results of this. The top plot shows the return at every single iteration. Human demonstration set 1 and 3 are omitted in this plot for readability. Taking a closer look, it can be seen that with pure Q-learning the algorithm needs around 150 iterations to converge towards a satisfactory policy. For the human demonstration set 1, the algorithm converges straight away. Human demonstration set 4 shows poor behavior, which shows that such a bad set of human demonstrations does not improve anything.

In the lower plot of the figure, the average return is plotted for the different demonstration sets and the pure Q-learning. It can be seen that set 1 and 2 show similar behavior, which is logical since the demonstrations are almost identical. Set 3 is a little bit worse, but still comparable to the first two sets. From this graph it can be concluded that using a proper human demonstration set, the average return will be high, since the algorithm converges straight away to a good policy, compared to pure Q-learning, that needs some more time. Furthermore, it can be seen that using a bad set of human demos yields bad results. Interesting to note here, is that for set 1 and 2 the greedy policy in the end results in a return of -4. This is higher than the return achieved in the demonstration runs themselves, which means that the algorithm does use the human demonstrations, but not blindly copies them, as it outperforms the human demonstrator.

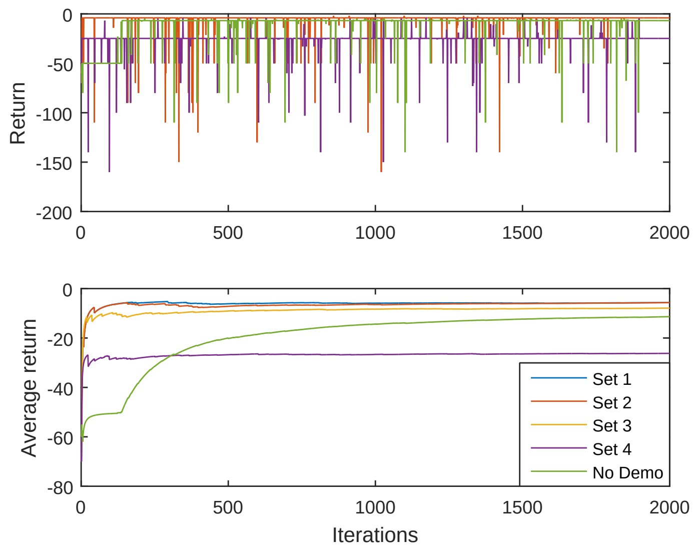


Figure 5-4: Return and average return for Q-learning and Q-learning with human demonstrations

Another interesting result can be seen in Table 5-4. Here the average return and the final greedy return are presented for Q-learning and the four human demonstration sets over 5000 iterations. For the Q-learning, the the Q-values are initialized at zeros here, to stimulate optimism in the face of uncertainty. From the table it can be seen that the Q-learning algorithm has, by far, the worst average return over 5000 iterations. The final greedy return, however, is best for the Q-learning. This solution has not been found by the algorithm using human demonstrations, since these are from the start on more focused towards solutions which are similar to the ones demonstrated by the human. The solution found by the algorithm using human demonstrations results in a sequence of states visualized in Figure 5-5. The action sequence which yields the -1.96 score, which is presented in Figure 5-6, shows that the quadrotor speeds up to the right, and does not make the load swing up to the right, but to the left. This non-trivial solution was not part of the human demonstrations and differs too much, to be found by the human demonstration algorithm.

Table 5-4: Average and greedy result

Set number	Average return	Final greedy return
1	-5.15	-4
2	-5.45	-4
3	-7.37	-5.76
4	-25.60	-24.84
Q-learning	-40.75	-1.96

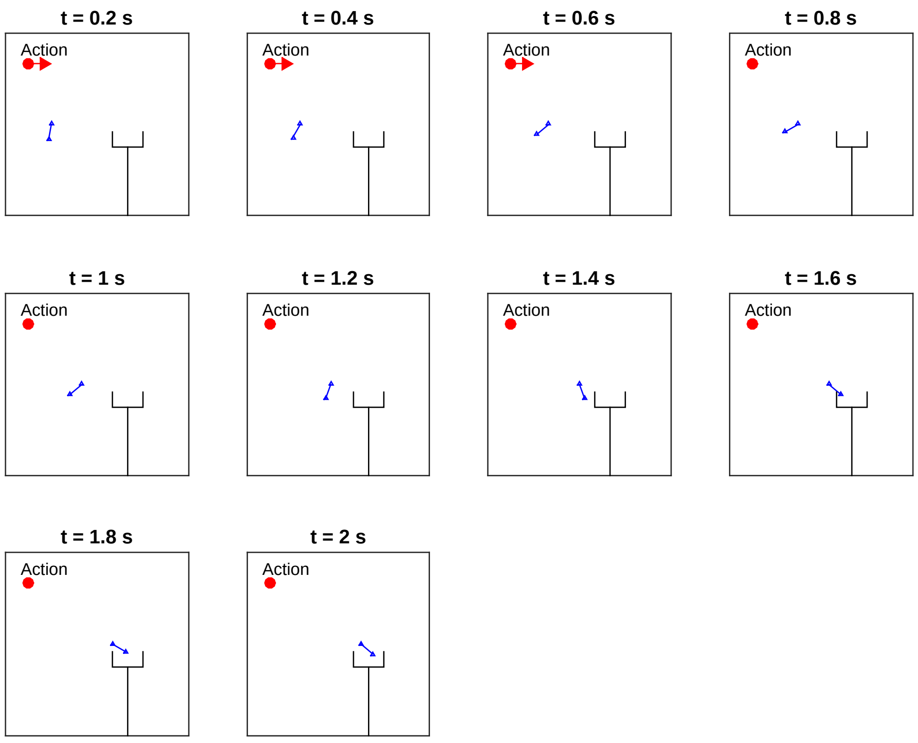


Figure 5-5: Snapshots of the solution found by the Q-learning algorithm using human expert demonstrations

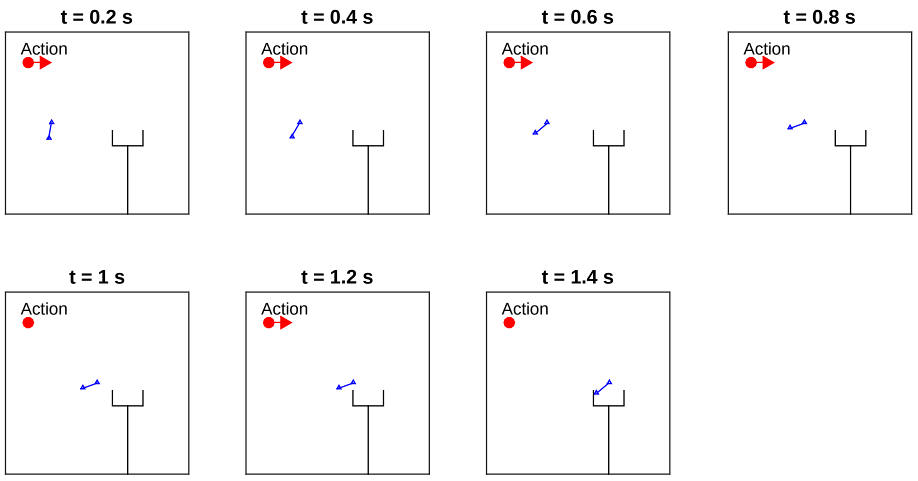


Figure 5-6: Snap shots of the solution found by the Q-learning algorithm

5-4 Value function analysis

To see whether the discrete values learned by the Q-learning algorithm make sense, and to see whether some useful relations can be found to use in basis functions, the values obtained in the discrete 2D simulation are studied. An important aspect of the discrete nature of the simulation so far is that a lot of state action combinations have never been visited, and as such have a zero value.

5-4-1 Number of visits and value representation

When looking at the number of visits, two things are interesting to study. Firstly, the total number of visits can be studied. This depends on the number of iterations performed. Comparing this number to the total number of state-action combinations, is a measure of relative visitation. The total number of state action combinations can be found looking at the discretization (Table 5-5). The used discretization results in 876,645 state-action combinations.

Table 5-5: Discretization of the states of the system

	minimum	maximum	spacing	number of entries
x_q [m]	-2	8	0.5	21
$\dot{x}_q$ [m/s]	-5	5	1	11
θ [deg]	-110	110	10	23
$\dot{\theta}$ [deg/s]	-540	540	20	55

For the analysis of the value distribution in the discrete simulation, 100,000 episodes are performed. This results in a total of 1,422,657 visits. This means that on average every state-action combination is visited 1.62 times. This is a low number, but because of computational limitations, the number of 100,000 episodes was deemed a reasonable maximum. The reason that with so few iterations, still a good policy was found, can be attributed to the fact that a large portion of the state-action space was never visited. Studying the results, it was seen that only 285,649 state-action combinations were ever visited, which is 33% of the total number of state-action combinations.

To reach this percentage of visited state-action combinations, one adjustment had to be made in the algorithm. Before, the initial conditions were the same for each iteration, i.e., $x = 0$ m, $\dot{x} = 0$ m/s, $\theta = 0$ deg, $\dot{\theta} = 0$ deg. Using always the same starting point results in a low number of visited state-action combinations. Only 69,379 state-action combinations are visited in that case, which is only 8% of the total. To solve this issue, the initial values of x and θ were randomly selected each iteration. The number of visited state-action combinations would have even be larger when $\dot{x}$ and $\dot{\theta}$ were randomly initialized as well. However, in this case the problem becomes a lot harder to solve for the algorithm, because of the low similarity in initial condition, and as such needs a lot more iterations to yield a good policy. Because of the limited computational resources only x and θ were initialized randomly.

Since the value matrix has five dimensions for this problem, a comprehensive graphical representation requires some creativity. In this section four plots are presented which average the value over all state-action combinations which have the same value for one of the parameters.

Figure 5-7 for example, shows the average over all state-action combinations with at a certain x-position. In the upper plot, the number of visits is plotted, and in the lower plot the mean value is plotted.

It is important to note however, that to make the plots more useful, the value is plotted for five cutoff numbers (C). When the cutoff number is 5 for example, all state-action combinations that have less than 5 visits, are neglected. This operation is included, since it shows the values more distinct. Since the learning rate is rather low in this algorithm (0.01), the value is changed only little every visit. This means that the value only converges after a lot of visits, and as such it could be argued that state-action combination with a low number of visits do not give a good representation of the true state value. Setting the cutoff number to high however, would mean that only a few state-action combinations remain to take the mean of, which results in poor results as well.

5-4-2 Value function results

Figure 5-7 shows the value as a function of position. First of all it can be seen that the number of visits is relatively equally spread. The only exception on this are the edges of the domain, where the quadrotor can get stuck. The value shows the expected behavior. The value is highest between $x = 4$ and $x = 6$, which is the location of the goal. Regarding the trend in this plot, the distance to goal seems a gradual relation with the value.

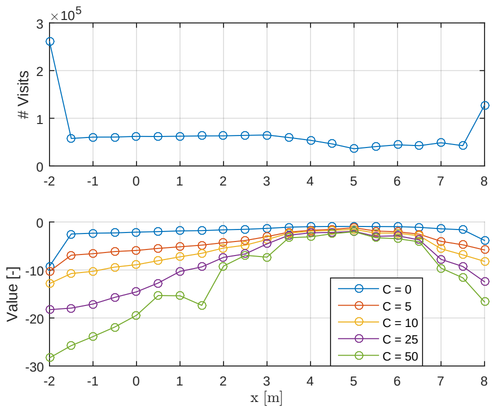


Figure 5-7: Accumulated value as a function of position obtained from the preliminary 2D simulation

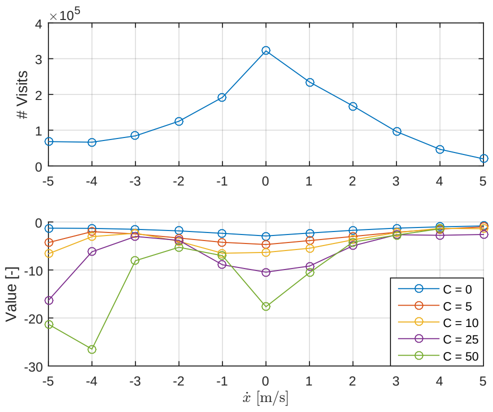


Figure 5-8: Accumulated value as a function of velocity obtained from the preliminary 2D simulation

In Figure 5-8 the value as a function of the velocity is plotted. In the upper plot it can be seen that the number of visits is highest at $\dot{x} = 0$, which is attributed to the fact that this is the initial condition, and as such is the starting point for each iteration.

Looking at the value plot, it can bee seen that the value is increasing for increasing positive velocities. This makes sense, since it means that heading faster to the goal yields a higher value. Zero velocity corresponds to a low value, which is logical. Moreover it can be seen that negative velocities, when moderate, yields quite high values as well. This can be explained by the fact that this is necessary when the starting point is to the right of the goal. Because of the layout of the problem, the region right of the goal is a lot smaller than the region left of the goal, and as such, in general, a lower velocity to the left is wanted.

Figure 5-9 shows the value as a function of slung angle. For this slung load task, the load should be swung up to make the load reach the goal. In the plot it can be seen that this behavior is reflected by the value function. Slung angles between -20 and 20 degrees have a low value, since at this angle the load would be not high enough to reach the goal. Slung angles with a magnitude larger than 80 degrees show low values. This behavior can be explained by the big penalty which is given at absolute angles of 90 degrees and larger.

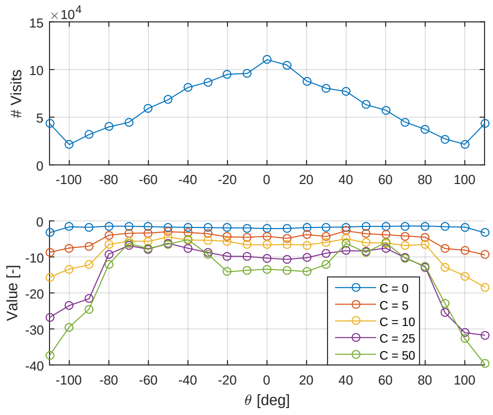


Figure 5-9: Accumulated value as a function of slung angle obtained from the preliminary 2D simulation

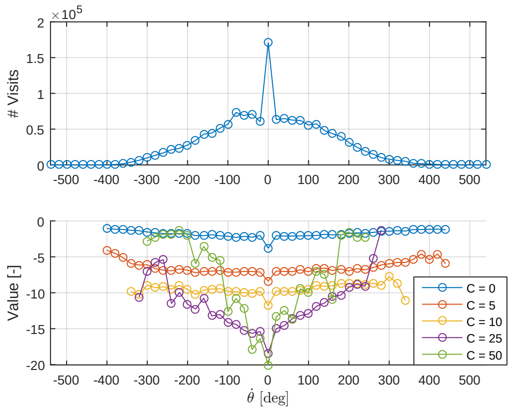


Figure 5-10: Accumulated value as a function of slung rate obtained from the preliminary 2D simulation

The final plot to study is Figure 5-10. In the number of visits it can be seen that low angular rates are visited relatively often. The peak at zero angular rate can be attributed to the fact that each iteration starts at zero angular rate. In the value, logical conclusions are hard to deduce. Purely looking at the values, it seems that higher angular rates are higher valued. This should, however, be probably attributed to the fact that the number of visits is higher at low angular rates, and the values did not converge yet.

From these results it can be concluded that the values show logical behavior, that matches the expectation. It is hard to conclude though that these results can be a foundation for good basis functions, since no coupling effects are included here. The basis functions which could be deduced from these results are: distance to goal, velocity towards goal and a basis function which rewards slung angles between 30 and 70 degrees, either positive or negative. For the angular rate, no clear pattern could be found. The combination of the aforementioned basis functions were applied to this problem, but do not show satisfactory results. It can be concluded that more sophisticated basis functions are needed for a good continuous value approximation.

5-5 Conclusions on the preliminary simulation

Looking at the difference between the pure Q-learning (benchmark algorithm) and the algorithm using human demonstrations, a number of conclusions can be drawn. First of all, it is shown that using human demonstrations results in a fast, almost immediate convergence towards a satisfactory policy. Furthermore, it is seen that the resulting policy is better than the demonstrations by the human. It must be stated though, that the quality of the human demonstration is an important factor regarding the performance of the algorithm. When the human demonstration is not good at all, the algorithm will not benefit, and could even perform worse than pure Q-learning, which uses no prior knowledge at all. A final remark is that, when the algorithm uses human demonstrations, it can be the case that the optimal policy will not be found, especially when this optimal policy is completely different from the (sub optimal) policy deducted using the human demonstrations. In short, from this 2D simulation, it can be concluded that human demonstrations drastically improves the time to convergence, but can result in not finding the optimal policy when this optimal policy is too different from the demonstrations given by the human expert.

Chapter 6

Plan of Approach for the Research

In the previous chapter a 2D, discrete state simulation showed promising results. In order to assess the performance of the selected algorithms in a more realistic setting, a plan of approach is formulated in section 6-1. The most important next steps are the evaluation of the algorithm in a 3D setting and, if that shows promising results, a real life experiment on a quadrotor. Section 6-2 gives an overview of the 3D simulation plan, after which section 6-3 presents the experimental setup.

6-1 General plan of approach for the research

Following from the conclusions drawn from the literature study, a plan of approach for the remainder of this research is formulated. As was mentioned in the conclusion of the literature study, a good implementation of human expert demonstrations within a reinforcement learning algorithm for quadrotor control, which is able to adapt to changing conditions, is not yet available. As such, the main goal is to construct such an algorithm. As a basis for this, the approach of Shaker [15] will be used, since this approach is model free and online, and shows the opportunity to be extended with the use of human demonstrations.

In addition to the previously defined research questions in the Introduction, the following more specific questions should be answered in the remainder of this research:

- How does the proposed algorithm compare to Q-learning in a 2D simulation?
- How does the proposed algorithm compare to Q-learning in a 3D simulation?
- Does the proposed algorithm perform on a real quadrotor experiment?
- What are the requirements for a set of human demonstrations to be useful for the proposed algorithm?

Figure 6-1 gives an overview of the research project, progress flows from top to bottom. Up until this point, the literature study has been performed based on the research questions. A basis for an algorithm has been selected following from this research.

First a preliminary two dimensional simulation will be carried out to evaluate the potential of the proposed approach, this simulation is finished for discrete state, and will be described in the next chapter. The 2D simulation will be extended to function in continuous state. After the 2D simulation, the algorithm should work for continuous state and will be tested in a 3D simulation, fulfilling the benchmark task of a slung load drop off. When the algorithm shows good results in this simulation, the algorithm will be tested in an experiment on the Parrot AR Drone. Between these steps, the algorithm and its settings will evolve, and as such it is likely that every now and then a step back should be taken, which in the figure is indicated by the arrows.

Within all steps of the research (except for the experiment), the performance of the algorithm will be compared to the performance of a reference reinforcement learning algorithm, which will be Q-learning. The comparison will be based on the performance of the greedy policy after a certain specified time. Next to this, the average return over all learning iterations will be compared, as well as the time to converge to a satisfactory policy. In this case the policy is referred to as "satisfactory" when the slung load will be dropped off at the goal, however, using the fastest route is not required.

Within the algorithm, the effect of different sets of human demonstrations will be evaluated. The number of trials, the variance in the trials, and other variations between different sets of human demonstrations, will be studied to find guidelines for the used of human demonstrations.

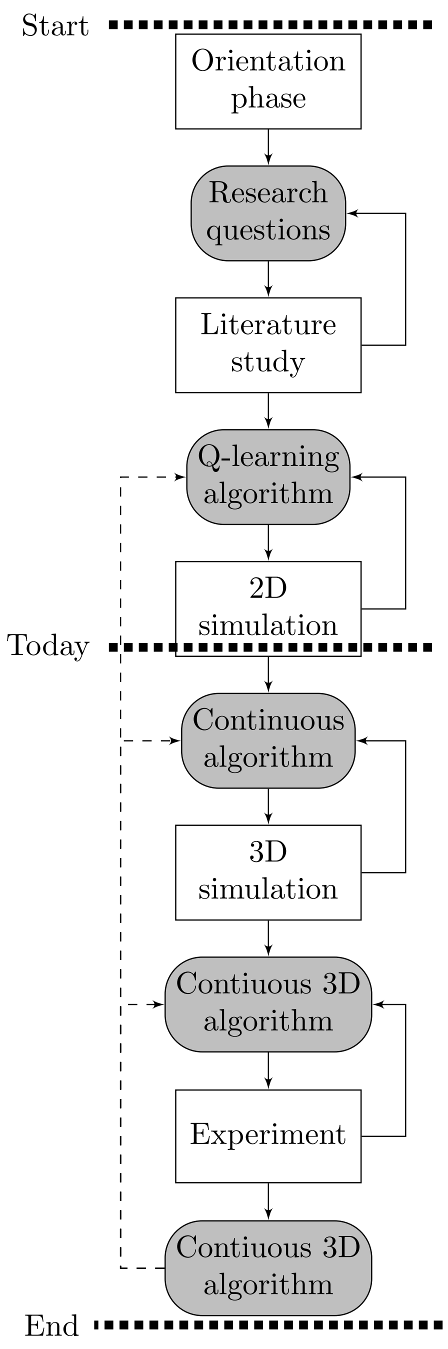


Figure 6-1: Research overview

6-2 3D Simulation Plan and Architecture

In this simulation, the quadrotor will have all six degrees of freedom, and the slung load will have either two or three degrees of freedom. When only a taut cable will be modeled, the position of the slung load can always be described by two swing angles. If a situation where the cable is slack will also be modeled, the system can switch to three degrees of freedom; the location of the load in three axes. Whether this will be modeled, is to be determined during the setup of the 3D simulation.

The task itself will also be extended to a 3D task. This means that the goal basket should be reached using lateral and longitudinal control, and the goal will be at another altitude than the starting point of the quadrotor. The initialization position of the quadrotor will be varied, to make the algorithm flexible.

For the 3D simulation, the following components should be in place:

1. 3D Dynamics model of a quadrotor (Poppe's model [27] - Simulink model including the dynamics of a quadrotor which was used for an nonlinear dynamic inversion controller. The model differentiates between dynamics in small attitude angles and large attitude angles)
2. Implementation of the slung load in this 3D model
3. Low level controller allowing to control the quadrotor with a joystick. To determine to which degree the low level control must be implemented, an interview will be conducted with an experienced quadrotor pilot, so that controlling the quadrotor in simulation will match controlling the quadrotor in real life.
4. Real time (RT) 3D simulation environment for pilot to control the quadrotor (simulink)
5. Generation of useful sets of human demonstrations.
6. Reward structure
7. Implementation of the RL algorithm in 3D
8. optional: Noise realization to model wind, sensor noise etc.
9. A value approximation method (Basis functions)
10. A non-real time simulation to make iterations for the RL algorithms

An overview of the intended architecture is presented in Figure 6-2, showing how the different bullet points from the list above interrelate. The 3D dynamics model (1) is at the core of the simulation. It outputs, in combination with the slung load block (2), the state of the vehicle. The state is fed back to the controller (3), which also gets the action input. The controller block uses a control law to calculate the required rotational velocities, which are the input for the 3D model. The actions can be generated by either a pilot (5), or the reinforcement learning algorithm (7). For the pilot to control the system, a visual feedback is required from the states of the vehicle in real time (4). The reinforcement learning block needs as input the reward, which is calculated by the reward structure block (6) and the state. To make the model even more realistic, wind and other disturbances can be included using the disturbance block (8). A disturbance could also be added to the state feedback, to model sensor noise, this is not included in this overview for clarity of the figure.

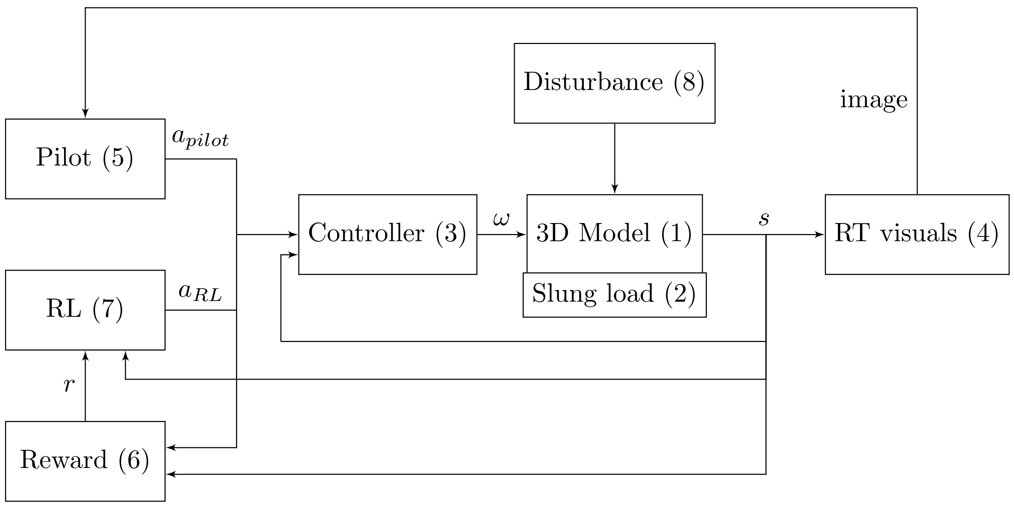


Figure 6-2: Overview of 3D simulation architecture

6-3 Experimental Setup

In the case that the reinforcement learning algorithm using human demonstrations still looks promising after the 3D simulation, an experiment will be carried out. This experiment will be done in the Cyberzoo [1] at the faculty of Aerospace Engineering in Delft, using a Parrot AR Drone 2.0 [2]. The task to be performed will be the same as in the 3D simulation.

6-3-1 Test setup

The experiments will be carried out in the Cyberzoo, which is located in the "Vliegtuighal" at the faculty of Aerospace Engineering. The Cyberzoo consists of a 10m x 10m x 10m area, which is enclosed by safety nets. Figure 6-3 gives an impression of the Cyberzoo. The Cyberzoo is equipped with an Optitrack [3] system, which provides a simulated GPS for indoors. This system is able to track the position and orientation of vehicles within the Cyberzoo, by the use of markers on the vehicles and cameras in the arena. Within the experiment, the quadrotor and the slung load should be equipped with markers, to determine their state at all times.

[1] TU Delft Robotics Institute. http://robotics.tudelft.nl
[2] Parrot AR.Drone 2.0. http://ardrone2.parrot.com/
[3] Optitrack$^{\text{TM}}$. http://www.naturalpoint.com

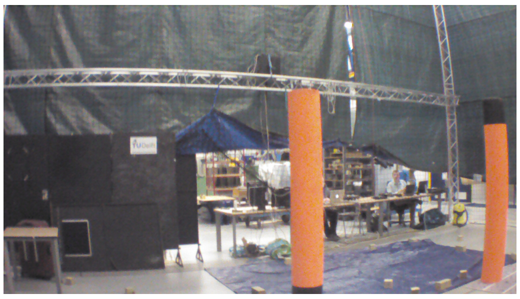


Figure 6-3: Cyberzoo flight arena

Figure 6-4: Parrot AR Drone 2.0 (picture from http://ardrone2.parrot.com/)

The quadrotor selected to implement the algorithms on, is the Parrot AR drone 2.0. This quadrotor is selected because it is available at the faculty of Aerospace Engineering, there is a lot of experience on it in house, and it is relatively easy to program. Figure 6-4 shows the Parrot AR Drone 2.0. It is equipped with a front and bottom camera and weighs 420 grams. The AR drone 2.0 has a 1 GHz 32 bit ARM Cortex A8 processor which runs Linux 2.6.32. For position and attitude estimation there is a gyroscope, magnetometer and accelerometer on board.

6-3-2 Implementation of the algorithm

The implementation of the algorithm on the AR Drone will be done using the open source software Paparazzi [4]. Within the Paparazzi library, a primary attitude controller should be selected as well as an altitude controller. As such, the quadrotor should be controllable by a human operator. When this low level control is in place, the reinforcement learning algorithms should be implemented, which uses the state estimation of either the Optitrack system, or the on board state estimation to calculate rewards. The output of the programmed algorithm should be a control input for pitch, roll and yaw, as well as thrust.

6-3-3 Human demonstrations

For the experiment, at least three good pilots have to be found in order to generate a number of human demonstration sets. Each demonstration set should consist of at least 10 trials. Each pilot will be asked to complete several sets of demonstrations. Some sets should be

[4]Paparazzi Free Autopilot. http://wiki.paparazziuav.org

performed when the pilot tries his/her best to finish the task. Other sets should include some deliberate deviation from the optimal strategy to a varying extent, in order to be able to compare different kind of demonstration within the reinforcement learning algorithm.

Chapter 7

Conclusion on the Preliminary Research

This preliminary research gives an insight into the use of human expert demonstrations to improve reinforcement learning for quadrotor control. The following research sub-questions are formulated for this research:

1. How can human expert demonstrations be used within RL algorithms?
2. How can the performance of a RL algorithm be defined?
3. How do RL algorithms using human expert demonstrations compare to each other, and how do they compare to existing techniques?

In the literature review, the first question is partly answered. It is found that in literature two ways of using human demonstrations within RL for quadrotor can be identified. The first one is using a set of human demonstrations to identify a model of the system, which is used offline to learn a policy. This method is model based and offline, and hence not easily extensible to more complex vehicles, and not able to adapt to changing conditions, which both are demands formulated in this research. The second way of using human demonstrations, is to use them within a least squares policy iteration algorithm, in order to have a good initial guess of a satisfactory policy. This approach can be used online and does not require a model. As such this approach fulfills the formulated demands for the algorithm, and hence will be pursued in the remainder of this research.

The second question is also studied within literature. Not many quantitative comparisons are found in literature. The most relevant performance metrics are found to be: Time to learn a good policy, performance of final policy and the accumulated return over the whole learning process. Using these metrics, both the quality of the learned policy, as well as the time it takes to develop it, are compared.

The third question is partly answered in the preliminary simulation chapter. A simple, discrete, 2D simulation shows that the use of human demonstrations within a Q-learning algorithm proves to be successful. Less time to learn is required to reach a satisfactory policy, and the accumulated return ends up a lot better than without using human demonstrations. It is found, however, that the final policy found using human demonstrations may not be the optimal policy. This effect has to be studied further, as well as the impact of different kinds of sets of human demonstrations.

In the next phase of this thesis the third question will be studied in an increasingly more realistic environment. At first, the 2D simulation will be extended to continuous state. Secondly, a 3D simulation environment will be created to evaluate the performance of the algorithm in three dimensions, while improving it in parallel. Finally, the algorithm will be tested on the Parrot AR Drone 2 in a real experiment. This should lead to a full and realistic answer to the third research question, thereby answering the main research question which is: "How can we use demonstrations of a human expert pilot to improve the performance of reinforcement learning for quadrotor control?"

Part III

Intermediate Research and Additional Results

Chapter 8

Balancing Pendulum Problem using Offline LSPI

In the preliminary research (Part II) Q-learning was applied to a 2D quadrotor slung load problem. To extend the concepts to the LSPI algorithm, and thereby allowing for continuous Q-function approximations, in this chapter the Offline LSPI [22] algorithm is implemented on a simple problem: balancing a pendulum. In this example, a pendulum which starts upright, has to be balanced as long as possible, with a small noise on the action. In this chapter this example is briefly described, and some of the results when using this example, with offline LSPI, are presented.

8-1 Problem description: balance pendulum

The balance pendulum example used in this chapter is exactly the same as used by Lagoudakis and Parr in their paper on offline LSPI [22]. As such, the problem description from their paper is copied here.

"The inverted pendulum problem requires balancing a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to. Three actions are allowed: left force LF (-50 Newtons), right force RF (+50 Newtons), or no force NF (0 Newtons). All three actions are noisy; uniform noise in [-10, 10] is added to the chosen action. The state space of the problem is continuous and consists of the vertical angle α and the angular velocity $\dot{\alpha}$ of the pendulum. The transitions are governed by the nonlinear dynamics of the system [28] and depend on the current state and the current (noisy) control a:

$$\ddot{\alpha} = \frac{g\sin(\alpha) - \kappa ml(\dot{\alpha})^2 \sin(2\alpha)/2 - \kappa\cos(\alpha)a}{4l/3 - \kappa ml\cos^2(\alpha)} \tag{8-1}$$

where g is the gravity constant ($g = 9.81$ m/s^2), m is the mass of the pendulum ($m = 2.0$ kg), M is the mass of the cart ($M = 8.0$ kg), l is the length of the pendulum ($l = 0.5$ m), and

$\kappa = 1/(m + M)$. The simulation step is set to 0.1 seconds. Thus, the control input is given at a rate of 10 Hz, at the beginning of each time step, and is kept constant during any time step. A reward of 0 is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ signals the end of the episode and a reward (penalty) of -1. The discount factor of the process is set to 0.95. We applied LSPI with a set of 10 basis functions for each of the 3 actions, thus a total of 30 basis functions, to approximate the value function. These 10 basis functions included a constant term and 9 radial basis functions (Gaussians) arranged in a 3 x 3 grid over the 2-dimensional state space. In particular, for some state s = $(\alpha,\dot{\alpha})$ and some action a, all basis functions were zero, except the corresponding active block for action a which was

$$\left(1,\ e^{\frac{||s-\mu_1||^2}{2\sigma^2}},\ e^{\frac{||s-\mu_2||^2}{2\sigma^2}},\ e^{\frac{||s-\mu_3||^2}{2\sigma^2}},\ldots,\ e^{\frac{||s-\mu_9||^2}{2\sigma^2}}\right)^T \tag{8-2}$$

where the μ_i's are the 9 points of the grid $\{-\pi/4,\ 0,\ \pi/4\}$ x {-1, 0, 1} and $\sigma = 1$. Training samples were collected in advance from "random episodes", that is, starting in a randomly perturbed state close to the equilibrium state (0, 0) and following a policy that selected actions uniformly at random. The average length of such episodes was about 6 steps, thus each one contributed about 6 samples to the set. The same sample set was used throughout all iterations of each run of LSPI. "

8-2 Analysis of LSPI as implemented by Lagoudakis

In the algorithm, a number of parameters can be adjusted. These are the number of samples, the variation within the samples, the discount factor, the termination criterion and the maximum number of iterations. In the paper of Lagoudakis, the maximum number of iterations to balance the pendulum is 3000. A score of 3000 hence is a fully successful policy. The following observations were made:

- **Number of samples:** As is expected, the higher the number of samples used, the higher the number of iterations the pendulum is balanced. This was already shown by Lagoudakis, in Figure 8-1. Table 8-1 shows the results obtained in this reproduction of the problem. It can be seen that the general trend is similar, and the order of magnitude as well. In this research only 10 different sample realizations were used, where Lagoudakis used 100. Furthermore, we evaluated the resulting policy two times, where Lagoudakis did this a 1000 times.
- **Variation within the samples:** The way the samples are generated is done as in Lagoudakis' paper. Starting in a perturbed state around the equilibrium state (0,0), a random policy is followed. The degree of perturbation was varied to study its impact. It was seen that the larger the initial perturbation, the higher the variation within the samples, and thereby the better the performance. However, when the perturbation is too big, the pendulum is too far off the equilibrium position, and thereby the performance deteriorates. This behavior can be seen in the results in Table 8-2. For the calculations from now on, the maximum number of samples is changed to 600, to save computation time.

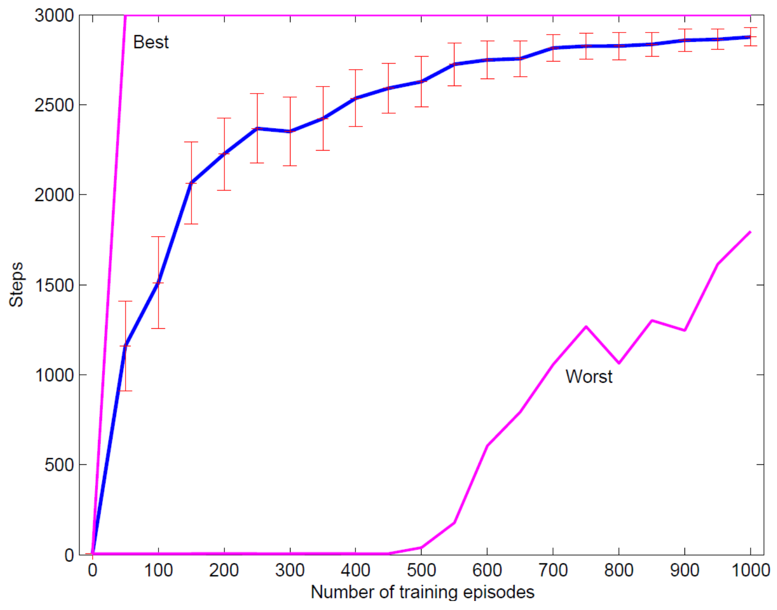


Figure 8-1: Inverted pendulum (LSPI): Average balancing steps, figure used from [22]

Table 8-1: Number of samples and performance

Number of samples	Avg. Balancing steps
10	322.9
50	918.0
100	1215.7
200	1808.2
500	1233.1
1000	2339.5

Table 8-2: Variation in samples and performance

Deviation	Avg. Balancing steps
0.001	194.4
0.01	351.0
0.1	321.7
0.5	72.2
1.5	33.3

- **Discount factor:** The default discount factor is 0.95. It was seen that using a value of 1 (no discount) yields a deficiency in solving for the Γ and Λ matrices, and hence gives no reliable results. It was seen that with values of γ between 0.95 and 1, the algorithm has problems converging, yielding poor results. Values between 0.85 and 0.95 show faster convergence. However, the resulting policy becomes worse with lower discount factor. It can be concluded that $\gamma = 0.95$ is a good choice for this problem.
- **Termination criterion:** The default value is 1e-5. At each iteration, the norm of the

weight vector is compared with the norm of the previous weight vector. When this value drops below the termination criterion, the algorithm stops. It was found out that as long as this criterion is small enough (< 0.1), it does not influence the outcome.

- **Maximum number of iterations:** This parameter determines the maximum number of iterations performed by the algorithm while not converging. Generally, a higher maximum number of iterations yields better results. It seems that when the number of iterations is at least 20, there is not much difference in outcome. It was seen that in general, the policy either converges within these 20 steps, or does not converges at all.
- **Number of RBFs:** The performance is better with more RBFs: an average of 481.3 balancing steps for 4 RBFs per state variables per action (51 basis functions in total) compared to an average of 350.95 balancing steps for 3 RBFs per state variable (30 basis functions in total).

8-3 Use of human demonstrations in offline LSPI for the inverted pendulum

The LSPI algorithm has been adjusted such that it can also use a set of Human Demonstration (HD) samples as input instead of randomly generated samples. In this section the results using the algorithm with different types of sample sets are presented. In Table 8-3 the results are presented for different human demonstration sets, as well as for two randomly generated sample sets, of which one was found to perform well, whereas the other was performing poorly. These randomly generated sets were included to demonstrate the range of performance that random sample sets can get. Furthermore, the table presents the result for the setting mentioned in the previous section: at each execution of the algorithm a different randomly generated sample set was used.

Table 8-3: Human demonstration sets and performance

Sample size	Sample type	Avg. Balancing steps	Remark
1000	HD	128.8	
976	HD	126.7	
1976	HD	108.7	Merged two first sample sets
464	HD	11.4	
999	random	49	Poor random sample
1032	random	600	Good random sample
variable	random	351.0	Previously used initialization

Looking at this table allows to draw some conclusions. Firstly, looking at the lower three sample sets, it can be concluded that there is a high variance between different random sample sets. This means that the algorithm (for this number of samples) is sensitive to the sample set used. This fact puts a large disclaimer on all the found results: A large sweep of different random seeds should be used to give a better analysis, since there is a large spread between different random seeds. Looking at the human demonstration results, it can be seen that in general the human demonstration sample sets provide inferior results compared to the randomly generated samples. Some tricks were tried to trim the human demonstration sets, only using “useful” samples. This did not yield better results, and it is hard to determine what actually is a “useful” sample.

It can also be seen that apparently, combining two human demonstration sample sets does not yield a better result. This is a weak conclusion as well though, as it was found out that the initialization of the weight vector already results in a large difference in found policies, so here again a larger study should be carried out to give significant results.

A clear difference between the randomly generated and the human demonstration samples can be seen in Figures 8-2 and 8-3. The variance in the samples is bigger for the random generated samples, this can be attributed to the fact that the random samples are including a bigger part of the state space since they are generated using a random policy. This behavior can be clearly seen as well in Figure 8-4.

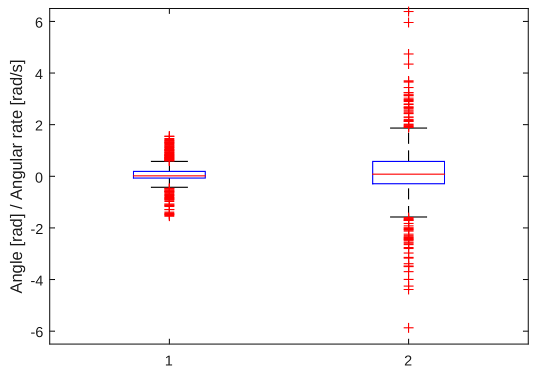


Figure 8-2: Human demonstration samples. Left: angle, right: angular rate

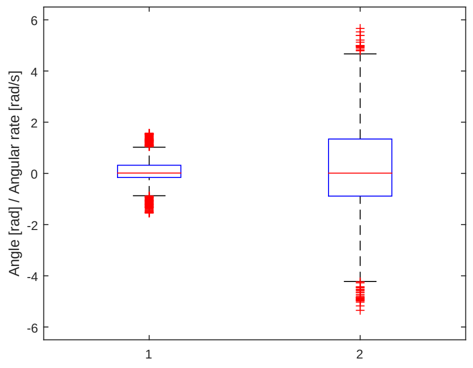


Figure 8-3: Random simulator generated samples. Left: angle, right: angular rate

8-4 Observations and remarks

The following observations and remarks can be made studying the offline LSPI algorithm on this balance pendulum problem:

- Different initial weight vectors already yield highly different results: larger numbers of evaluations are needed for significant conclusions.
- More RBFs are beneficial for the outcome of a HD sample set (363.95 for hd976 with 4 RBFs, even 486.65 with max iterations on 50), this could be due to the fact that with HD sample sets the spread is a lot smaller in data points, and hence more subtle differences are to be identified.

  As we did not find satisfactory results for the offline algorithm on this problem, we decided that a problem which can benefit more of the understanding of a human, would probably be suited better for this research. As such a swing up pendulum was chosen, since this problem needs some understanding on future states, to be able to make it swing up. With the knowledge obtained in that problem, the results found in this

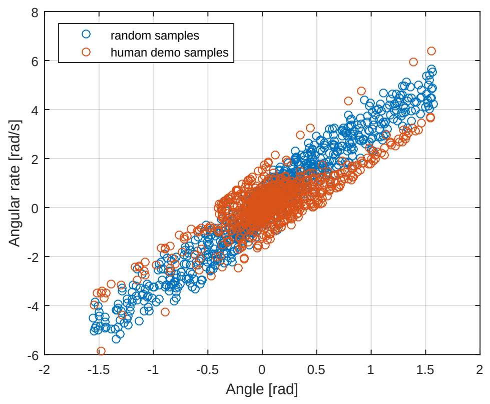


Figure 8-4: Spread of samples

chapter make sense. Using the human demonstration sample set, which is a lot less diverse than the random generated set, yields inferior performance. In the swing up pendulum we saw as well that a non-diverse human demonstration set does not form a good sample set to learn a Q-function from.

Chapter 9

Additional Research on the IOLSPI Algorithm

In Part I the Informed Online Least Squares Policy Iteration (IOLSPI) was presented and discussed. Some aspects of the algorithm and the way in which the results are statistically compared, were left out, to keep it concise. For the interested reader, this chapter provides more background on some aspects of the IOLSPI algorithm. In section 9-1 a more elaborate discussion of the implementation of the Radial Basis Functions (RBFs) in the algorithm is given. The addition of the forget factor in the algorithm will be studied more closely in section 9-2. The chapter ends with section 9-3, which gives insight on the statistical analysis that was used in processing the results in this thesis.

9-1 Radial Basis Functions implementation

An important aspect of the LSPI algorithm is the choice of Basis Functions (BFs) to approximate the value function. Many methods can be used to choose these BFs [11]. In this research RBFs are used, since they do not need a lot of hand-tweaking and generally provide good value function approximations. In Part I the RBFs are quickly introduced. The following sections give a more elaborate description of the implementation of the RBFs in this research.

9-1-1 The standard RBF

For the swing up pendulum problem an equidistant grid of 11x11 RBFs is used to approximate the value function. This means that the vector of RBFs is $\bar{\phi}(s) = [\phi_1(s), \ldots, \phi_{121}(s)]^T$. To make state-action BFs, the RBFs are repeated for the three different actions ($a = \{-3, 0, 3\}$ V, corresponding to a clockwise, counterclockwise or zero moment). The RBFs are evaluated for the current action and zero for the RBFs associated with the other actions. As such the vector of 121x3 state-action BFs can be represented as follows: $\phi(s, a) = [\mathcal{I}(a = -3) \cdot \phi^T(s), \mathcal{I}(a =$

$0) \cdot \phi^T(s), \mathcal{I}(a=3) \cdot \phi^T(s)]^T$. Here the indicator function $\mathcal{I}$ is 1 when its argument is true, and zero otherwise.

The radial basis function is the following:

$$\phi(s) = e^{-\frac{||s-\mu||^2}{2\sigma^2}} \tag{9-1}$$

In this equation μ is the RBF center location, σ the RBF width parameter, which is fixed at the distance between two RBF centers.

9-1-2 Normalization of the RBF

In the rotational pendulum case, the number of dimensions of the RBF is two, since there are two states of interest, angle and angular rate. To cancel out the difference in order of magnitude of the different state parameters, a transformation is applied to the input state signals to normalize these signals in the range from -1 to 1. This is done using Eq. 9-2

$$\bar{s} = -1 + 2\frac{s - s_L}{s_H - s_L} \tag{9-2}$$

The following equation is used to calculate the normalized RBF:

$$\phi(s) = e^{-\frac{||\bar{s}-\bar{\mu}||^2}{2\sigma^2}} \tag{9-3}$$

Figure 9-1 shows graphically how the RBF centers are distributed, for the angle and angular velocity of the quadrotor example.

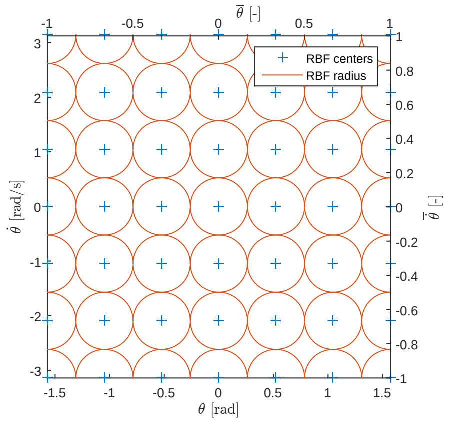


Figure 9-1: Location of the RBF centers, with their respective radii for the slung angle and angular rate of the quadrotor example. Top and right axis are the normalized scales.

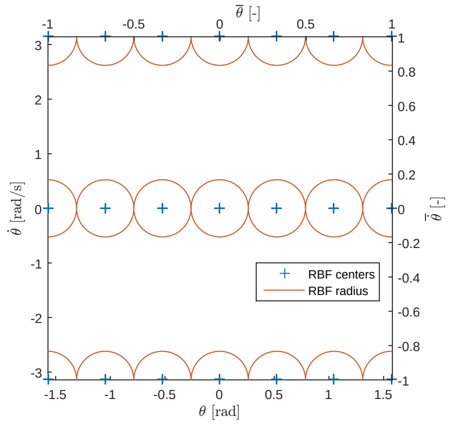


Figure 9-2: Location of the RBF centers, with their respective radii for the slung angle and angular rate of the quadrotor example with asymmetric spacing of the centers. Top and right axis are the normalized scales.

9-1-3 Asymmetric spacing of RBF centers

For the slung load problem the number of dimensions is four. As such, the grid of RBF centers will be four dimensional. To fit a good value function, that is able to follow the complexity of the function, enough RBFs should be used. Since it is expected that not each dimension has the same complexity, and as such needs the same number of RBF centers, an asymmetric grid of RBF centers will be used. Figure 9-2 shows the situation in which this is the case. This asymmetry gives rise to the problem that around $\dot{\theta} = -\pi$, $\dot{\theta} = 0$, or $\dot{\theta} = \pi$, the density of RBFs is a lot higher, than in the regions in between. The RBFs are summed, so a wavy pattern in the Q-function will emerge, with peaks around π, 0 or $-\pi$ and valleys in between. To alleviate this problem, a non-symmetric normalization is used. That is, the center locations as well as the states are normalized asymmetrically. For the center locations we have:

$$\bar{\mu}_x = [-1, 1] \tag{9-4}$$

$$\bar{\mu}_{\dot{x}} = \frac{c_{\dot{x}} - 1}{c_x - 1} \cdot [-1, 1] \tag{9-5}$$

$$\bar{\mu}_{\theta} = \frac{c_{\theta} - 1}{c_x - 1} \cdot [-1, 1] \tag{9-6}$$

$$\bar{\mu}_{\dot{\theta}} = \frac{c_{\dot{\theta}} - 1}{c_x - 1} \cdot [-1, 1] \tag{9-7}$$

$$\tag{9-8}$$

And for the normalized state we have:

$$\bar{\bar{s}} = [1, \frac{c_{\dot{x}} - 1}{c_x - 1}, \frac{c_{\theta} - 1}{c_x - 1}, \frac{c_{\dot{\theta}} - 1}{c_x - 1}]^T \cdot \bar{s} \tag{9-9}$$

In these equations c is the number of RBF centers in the respective dimension. This results in the situation depicted in Figure 9-3. The RBFs are ellipsoidal now, making sure that for each part of the state space the Q-function can properly be approximated and no wavy pattern as for the situation in Figure 9-2 will emerge.

9-1-4 Trimming the RBF centers: convex hull approach

In Part I it was mentioned that the execution time of the algorithm scales more than quadratic with the number of RBFs used. In this section an approach is introduced to limit the number of RBFs needed named the convex hull approach. This approach is not used in the results presented in this thesis, since further research on this method is needed to implement it in IOLSPI. The working principle of the approach is discussed in the remainder of this section.

Especially for higher dimensional problems, using an equidistant RBF grid, the number of RBFs grows quickly. However, in many problems a large part of the state space spanned by the RBF centers, is never visited because of physical limitations of the system. In the quadrotor slung load problem for example, the combination of a high slung angle, and a high angular rate of the slung load will never occur for this problem setup. As such it is a waste

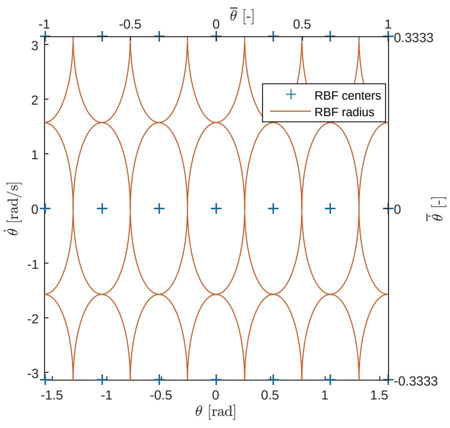


Figure 9-3: Location of the RBF centers, with their respective radii for the slung angle and angular rate of the quadrotor example with asymmetric spacing of the centers, and asymmetric normalization. Top and right axis are the normalized scales.

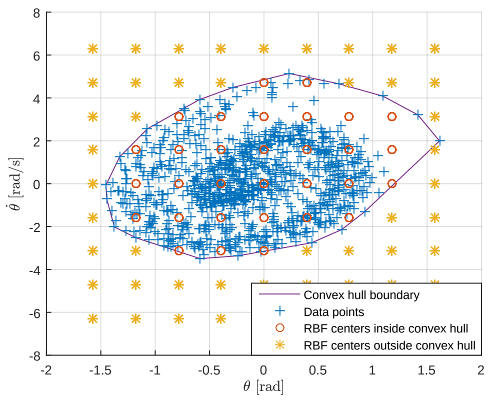


Figure 9-4: Location of the RBF centers for the slung angle and angular rate of the quadrotor example with a plotted sample set and the convex hull

of computational effort to include RBFs in these regions of the state space. To limit this issue, an approach is devised that gets rid of the RBF centers in the improbable regions of the state space. Figure 9-4 shows in two dimension what this method does. Using a sample set, obtained by a human demonstration or otherwise, a convex hull is determined. A convex hull of a set of data points is the smallest convex set containing all data points [29]. It is assumed that the Q-function only needs to be approximated in the reachable part of the state space. As such, all the RBF centers that are outside of the convex hull, are removed, since they predominantly effect the unreachable part of the state space.

Apart from the benefit of reducing the computational effort using this approach, from a data fitting point of view this approach proves itself useful as well. The RBFs which are outside the convex hull, are not forced to fit the local Q-function, since there is no data available at there region of influence. The RBFs inside the convex hull already fitted the actual Q-function. As such, the RBFs outside of the convex hull will fit the noise, and deteriorate the overal Q-function approximation.

This convex hull approach has not been applied yet for the results presented in this thesis, but shows a promising way of eliminating unused RBFs, thereby speeding up the algorithm, and improving the Q-function approximation.

9-2 Forget factor implementation

In Part I, in Algorithm 2 the implementation of a forget factor was mentioned. This section provides more insight in the way this forget factor is implemented, and some additional consideration regarding the forget factor.

Table 9-1: Forget factor and number of samples used

$N_{0.01}$	η
1000	0.997700
5000	0.999540
10000	0.999770
20000	0.999885
50000	0.999954
100000	0.999977
150000	0.999985
All	1

In the IOLSPI algorithm, the forget factor (η) is implemented as follows. Each time a sample is added to the Γ, Λ matrix or z vector, the previous version of these matrices and vector are multiplied with the forget factor. This way the relative importance of a sample decays exponentially with the decay factor. This can be seen in the following equations:

$$\Gamma_{k+1} \leftarrow \eta\Gamma_k + \phi(s_k, a_k)\phi^T(s_k, a_k) \tag{9-10}$$

$$\Lambda_{k+1} \leftarrow \eta\Lambda_k + \phi(s_k, a_k)\phi^T(s_{k+1}, \pi_l(s_{k+1})) \tag{9-11}$$

$$z_{k+1} \leftarrow \eta z_k + \phi(s_k, a_k)r_{k+1} \tag{9-12}$$

This means that after k time steps, a sample will have been reduced to a fraction of η^k of its original value. Depending on the forget factor, after a certain number of time steps, the weight of a sample drops below 1 % of its original value. The following equation calculates which forget factor corresponds to a number of samples $N_{0.01}$ that are still more than 1% of their original value.

$$\eta = 0.01^{\frac{1}{N_{0.01}}} \tag{9-13}$$

Table 9-1 shows a list of different values for $N_{0.01}$ and there corresponding forget factors.

As was mentioned before, each policy iteration step the modified Bellman equation is solved for the weights.

$$\frac{1}{n_s}\Gamma w^l = \frac{1}{n_s}z + \frac{1}{n_s}\gamma\Lambda w^l \tag{9-14}$$

Each term includes the scaling term $\frac{1}{n_s}$ which is included for numerical stability [22]. When the forget factor is included, the magnitude of the entries of the Λ and Γ matrices is different than without forget factor. As such it could be argued, that to keep a same order of magnitude in the entries of these matrices, the scaling should be adjusted as well. The following scaling is used:

$$\frac{1}{\sum_{i=1}^{k}\eta^i}\Gamma w^l = \frac{1}{\sum_{i=1}^{k}\eta^i}z + \frac{1}{\sum_{i=1}^{k}\eta^i}n_s\gamma\Lambda w^l \tag{9-15}$$

Having done some comparative tests between this scaling, and the original scaling, no significant difference was found, neither in execution time, nor in results. As such, for simplicity we stick to the original scaling.

9-3 Statistical analysis

This section gives a note on the way of using statistical analysis in obtaining the results for the (Informed Online) Least Squares Policy Iteration algorithm, which are presented in the paper (Part I), and in the following chapters.

Due to the fact that we use random sampling of the initial policy weights, and that the algorithm randomly selects an action when the Q-function has the same value for different action or when it is exploring, we sometimes find large differences in results while using the same algorithm settings, but for different seeds. To draw significant conclusions, a statistical analysis should be carried out on the different results. To do this, the algorithm is run for several independent random seeds. Using these different realizations, the 95% confidence interval can be calculated as well as the mean. The upper and lower confidence bounds are calculated using the following equations.

$$Y_{up} = \bar{Y} + \frac{1.96\sigma}{\sqrt{n}} \tag{9-16}$$

$$Y_{low} = \bar{Y} - \frac{1.96\sigma}{\sqrt{n}} \tag{9-17}$$

Where $\bar{Y}$ represents the mean, σ the standard deviation and n the number of realizations.

To obtain statistically good results throughout this thesis, the number of different random initializations needed to reach a good mean result with acceptable confidence bounds is studied. Figures 9-5 until 9-8 show the mean, 95% confidence bounds, and extreme values for one specific setting of the online LSPI algorithm applied to the rotational pendulum problem, for different numbers of realizations. In general it can be seen that the higher the number of realizations, the smoother the mean and the smaller the 95% confidence bounds. It is found that using 10 realizations is not enough. Up until 25 realizations, the confidence bounds become smaller and smoother. Between 25 and 50 realizations the difference is small, though still detectable. As can be seen below, using 100 realizations still gives some improvement. However, for computational reasons the standard number of realizations used will be 50.

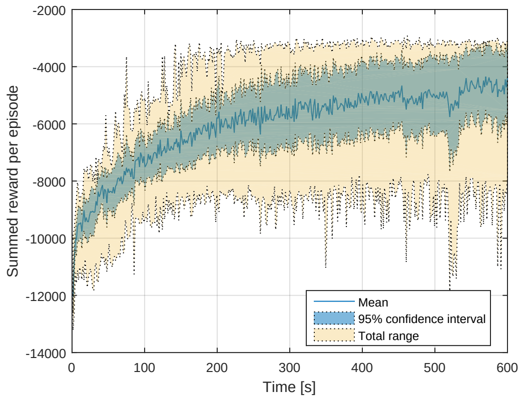


Figure 9-5: Online LSPI with 10 realizations: learning score as a function of time, $K_\theta = 100$, $\epsilon = 0.6$

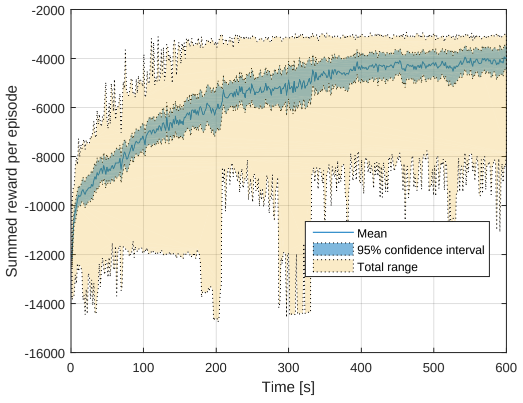


Figure 9-6: Online LSPI with 30 realizations: learning score as a function of time, $K_\theta = 100$, $\epsilon = 0.6$

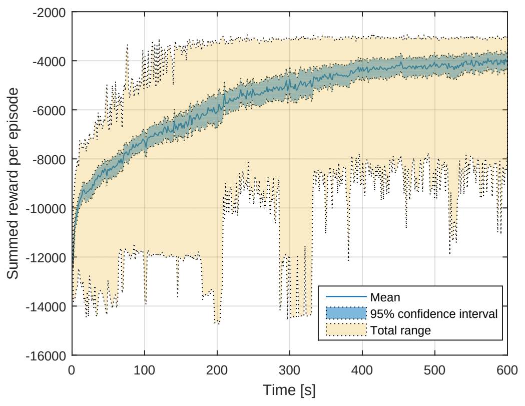


Figure 9-7: Online LSPI with 50 realizations: learning score as a function of time, $K_\theta = 100$, $\epsilon = 0.6$

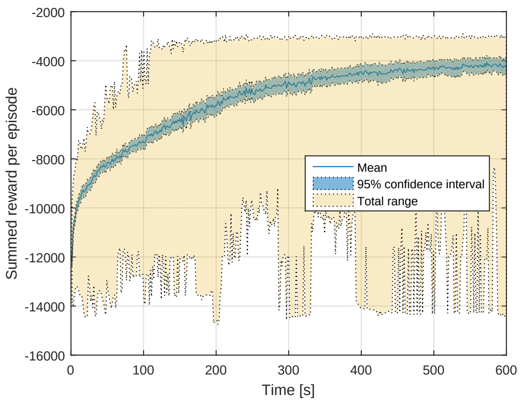


Figure 9-8: Online LSPI with 100 realizations: learning score as a function of time, $K_\theta = 100$, $\epsilon = 0.6$

Chapter 10

Rotational Pendulum Additional Results

This chapter is intended as an addition to section IV of the conference paper in Part I. The results presented here are for the rotational pendulum, as was described in the conference paper. This chapter is split in two sections: Section 10-1 presents a series of results for the online LSPI algorithm, which uses no pre-collected sample sets. Section 10-2 presents the results for the IOLSPI algorithm, mostly with the use of human demonstration sample sets.

10-1 Online LSPI

The reason to do a detailed study on the influence of different parameters on the online LSPI algorithm is that this allows for better understanding of the LSPI algorithm. Furthermore, it serves as a baseline for comparison with the IOLSPI algorithm, that uses human demonstrations.

10-1-1 Effect of tuning parameters for online LSPI

The effect of several tuning parameters will be studied. The influence of the update interval, exploration rate and discount factor will be analyzed.

Update interval

Figures 10-1 and 10-2 show the result of the online LSPI algorithm for different update intervals, varying from $K_\theta = 2$ (very optimistic), to $K_\theta = 5000$ (not so optimistic). Looking at the results it can be concluded that a too optimistic policy update does not yield a good score. This is attributed to the fact that when $K_\theta = 2$ for example. The policy will already be updated after two samples. Using only two samples to make the Q-function approximation is not enough. It can be expected that the policy which is found is rather arbitrary. Using this policy, the next two samples will be gathered, so these will not be representative either. As a result the policy never really becomes satisfactory. When K_θ is higher, the first policy update already steers towards a good policy, since the approximation of the Q-function is a lot better. As such the agent will learn a better policy more easily. It can be seen that for $K_\theta = 100$ and higher, the policy found is similar. $K_\theta = 1000$ is found to be best for this rotational pendulum problem. Additionally, it can be seen that a high update interval, 5000 for example, yields a little worse performance. This is attributed to the fact that the agent waits too long (5000 time steps) to improve its policy. As a result it takes longer to converge to a good policy, and thus, especially in the beginning, the performance is worse than for lower K_θ .

A final note to make on the update interval, is that the execution time of the algorithm is highly dependent on the update interval. This makes sense, since solving for a new policy is a relatively expensive operation within the algorithm.

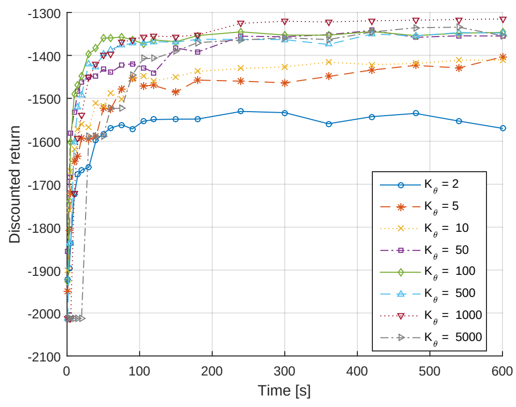


Figure 10-1: Online LSPI: Discounted return as a function of time for different policy update intervals, $\epsilon = 0.2$, using "hanging down ICs"

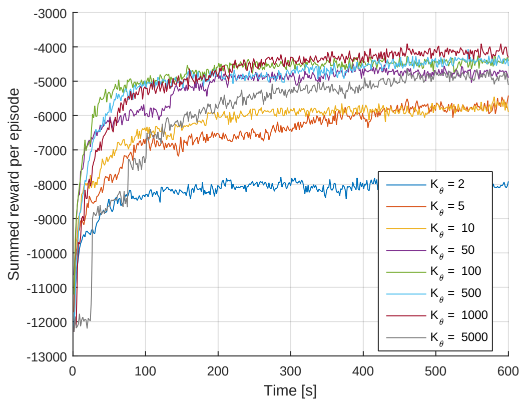


Figure 10-2: Online LSPI: Learning score as a function of time for different policy update intervals, $\epsilon = 0.2$, using "hanging down ICs"

Exploration rate

In the conference paper in Part I the results for different constant exploration rates was already presented. Figure 10-3 and 10-4 show in addition the result for different exploration decay factors, all starting with a fully exploratory policy ($\epsilon = 1$). The results found are not too different from the results for constant exploration rates. For the higher decay factors (0.9996 and 1) it can be seen that too little exploitation does not yield a satisfactory policy for this problem. For the decay factor of 0.9996 it can be seen that after some time (100 s) it becomes more exploiting, and as such the performance starts to go up. Looking to the summed reward per episode (Figure 10-4), it can be seen that the lower the exploration decay factor, the faster the learning score goes up. This is because the exploration factor decays, and as such the agent becomes more exploiting faster. A too low decay factor is not good either, since this results in too little exploration at the start.

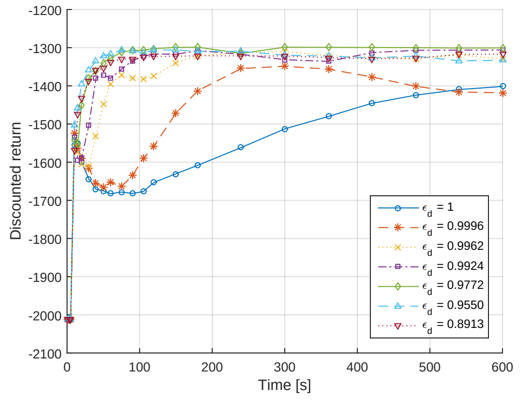


Figure 10-3: Online LSPI: Discounted return as a function of time for different exploration decay factors, $K_\theta = 1000$, $\epsilon_0 = 1$, using "hanging down ICs"

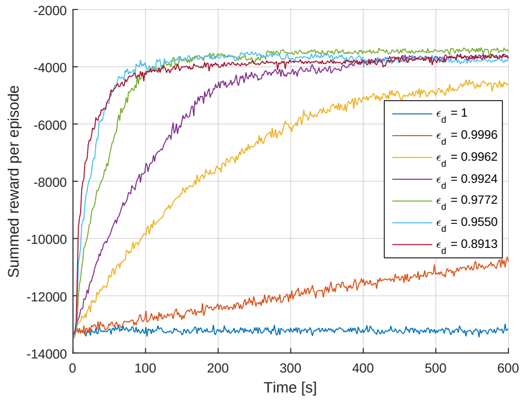


Figure 10-4: Online LSPI: Learning score as a function of time for different exploration decay factors, $K_\theta = 1000$, $\epsilon_0 = 1$, using "hanging down ICs"

Discount factor

The third tuning parameter which was varied is the discount factor. Figure 10-5 shows the discounted return for different discount factors. This plot is useless for comparison, since the discount factor determines for a large part the discounted return which is plotted. As a result it is much more interesting to study Figure 10-6, which shows the learning score for different discount factors. It can be seen that for a low discount factor ($\gamma = 0.8$), the performance is not too good. This makes sense, since the horizon of the agent is small. The more advanced states, at which the pendulum would be upright, are too far away for this shortsighted agent. Figure 10-7 shows the time history of the angle for a discount factor of 0.8. It can clearly be seen that the controller is not able to swing up the pendulum and reach a zero angle. A too high discount factor ($\gamma = 0.999$) does not yield satisfactory results either. The time history of the angle for this high discount factor can be seen in Figure 10-8.

10-1-2 Influence of noise

In the conference paper, and in the results presented so far, a perfect world is assumed, with no noise in the system. In real life there will always be a noise component. As such the effect of this noise on the system is studied. There are two ways in which a noise is introduced. The first way is a noise on the action. This noise is included in the following way:

$$a_p = a + n_a \tag{10-1}$$

In this equation a_p is the perturbed action, a the unperturbed action, and n_a a zero mean Gaussian noise with standard deviation σ_a. Since the action is the only moment acting on the system this noise on the action can be seen as either actuator noise, or as system noise (an external disturbance acting on the system).

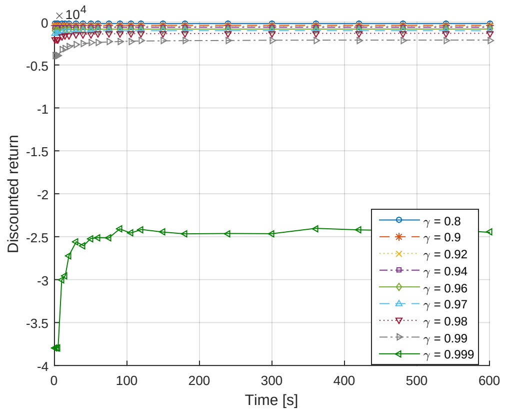


Figure 10-5: Online LSPI: Discounted return as a function of time for different discount factors, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

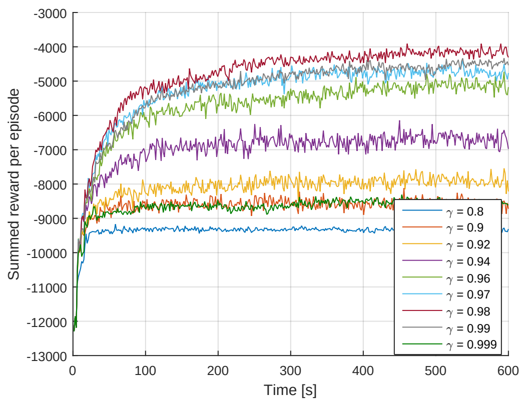


Figure 10-6: Online LSPI: Learning score as a function of time for different discount factors, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

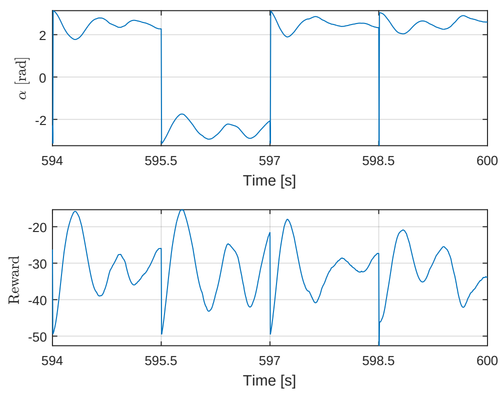


Figure 10-7: Online LSPI: Time history of angle and reward, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.2$, $\gamma = 0.8$, using "hanging down ICs"

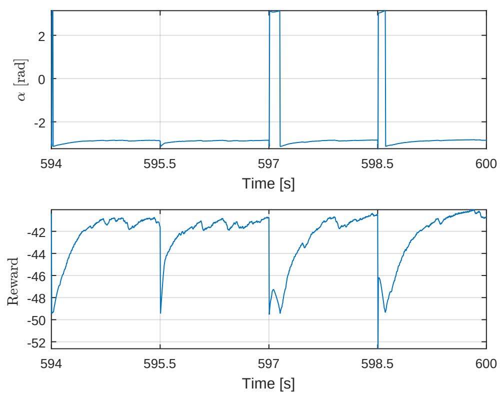


Figure 10-8: Online LSPI: Time history of angle and reward, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.2$, $\gamma = 0.999$, using "hanging down ICs"

The second source of noise is a sensor noise which is modeled. In real life, the state will never be measured exactly by the sensors; the measurement will always be perturbed a little. The measured state as such becomes:

$$s_p = s + n_s \tag{10-2}$$

In which s_p is the measured state, s is the true state and n_s is a zero mean Gaussian noise with standard deviation σ_s.

Noise on the action

Figures 10-9 and 10-10 show the results for four different action noise levels. It can be seen that including noise, the performance of the algorithm improves. This can be attributed to the fact that by introducing the noise, an implicit form of exploration is added. In this way the variety in samples becomes a lot bigger. As such especially the discounted return for evaluating the policies becomes better. Looking at the learning score, it can be seen that with noise the agent learns a bit quicker. However, in the end the higher noise (σ_a =10 V) shows a little lower converging score. This makes sense, since the control over the action is lower than without noise. However, looking at Figure 10-11, which shows the time history of reward and angle, it can be seen that the performance is still rather good: The controller can swing up the pendulum to a zero angle. The benefit arising is that the noise can also add to a faster convergence towards the upright position, since the absolute action might be bigger with the noise addition. This phenomenon can be seen in the last episode (from 598.5 s to 600 s), here the motor is able to swing up the pendulum in one go. However, the level of noise σ_a = 10 V is exaggerated, and not realistic.

It is assumed that a noise level with a standard deviation of 1 V is realistic. The plots show that the performance of the algorithm does not change too much. Hence it can be concluded that the algorithm also works well when actuator noise and external disturbances are present.

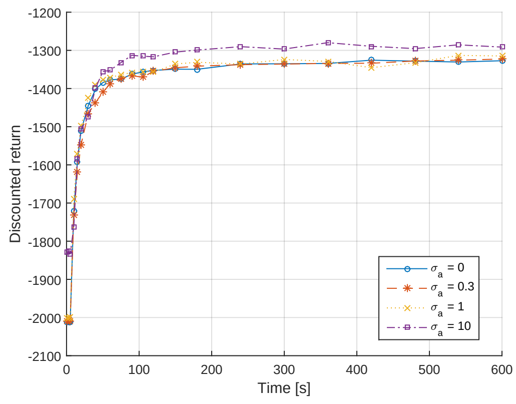


Figure 10-9: Online LSPI: Discounted return as a function of time for different action noise levels, K_θ = 1000, ϵ = 0.2, σ_a =[0 0.3 1 10] V, using "hanging down ICs"

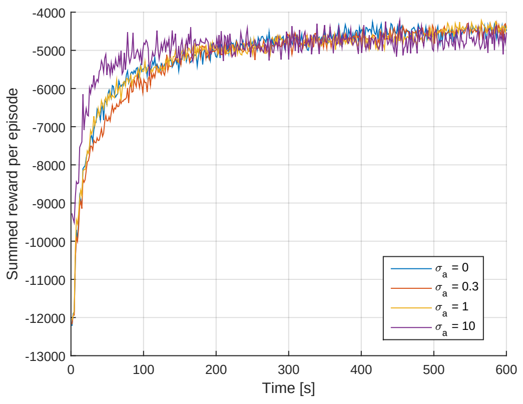


Figure 10-10: Online LSPI: Learning score as a function of time for different action noise levels, K_θ = 1000, ϵ = 0.2, σ_a =[0 0.3 1 10] V, using "hanging down ICs"

Sensor noise

In Figures 10-12 and 10-13, the results when the sensor noise is included are presented. In the figures it can be seen that a high sensor noise level ($\sigma_s = [25 \text{ deg}, \ 125 \text{ deg/s}]^T$) results in a reduction in performance. This result makes a lot of sense. It means that the agent selects the optimal action using a state measurement that is far off the actual state value. As such a large portion of the chosen actions is not optimal. It can be seen that for moderate sensor

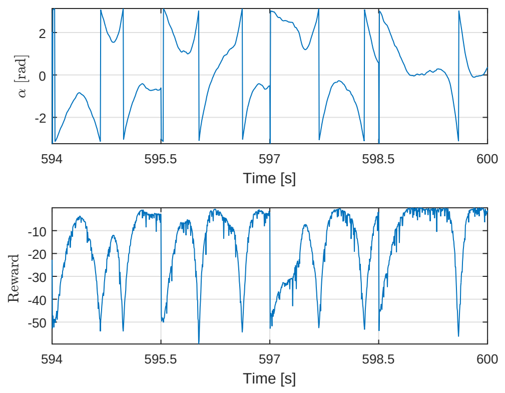


Figure 10-11: Online LSPI: Time history of the angle and reward, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_a = 10$ V, using "hanging down ICs"

noise levels the behavior is similar to the situation with no noise. It can be concluded that a moderate deviation in the sensor (2 degrees standard deviation on the angle measurement and 10 degrees standard deviation on the angular rate measurement) still yields good performance, hence the controller is robust to some sensor noise. It seems that the small noise level even improves the performance. This can be explained, similar to with the action noise, by the fact that it induces an implicit randomness in the algorithm, diversifying the sample spread. This diversification yields a better approximation of the Q-function.

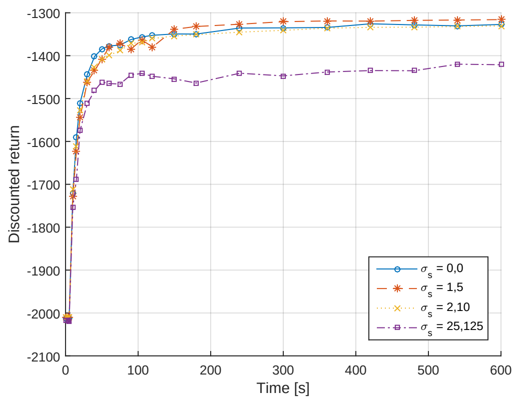


Figure 10-12: Online LSPI: Discounted return as a function of time for different sensor noise levels, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_s = \{0, 1, 2, 25\}$ deg and $\{0, 5, 10, 125\}$ deg/s, using "hanging down ICs"

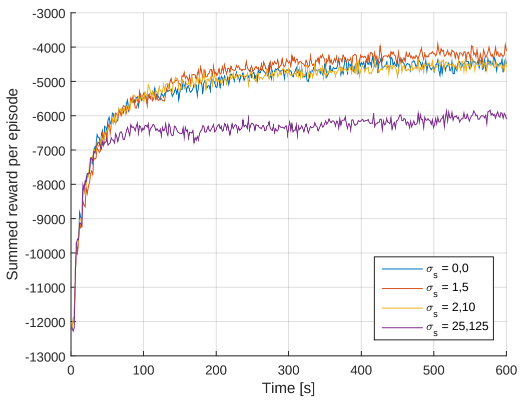


Figure 10-13: Online LSPI: Learning score as a function of time for different sensor noise levels, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_s = \{0, 1, 2, 25\}$ deg and $\{0, 5, 10, 125\}$ deg/s, using "hanging down ICs"

10-1-3 Behavior in changing conditions

As was mentioned in the conference paper in Part I, the dynamics of the system are changed externally, by altering the gravitational acceleration. We will study five different situations: $g = \{3.81, 7.81, 9.81, 11.81, 19.81\}$ m/s^2. Figures 10-14 and 10-15 show the learning behavior for these different situations, when they are constant, to see to what value they converge.

Sudden changes in conditions

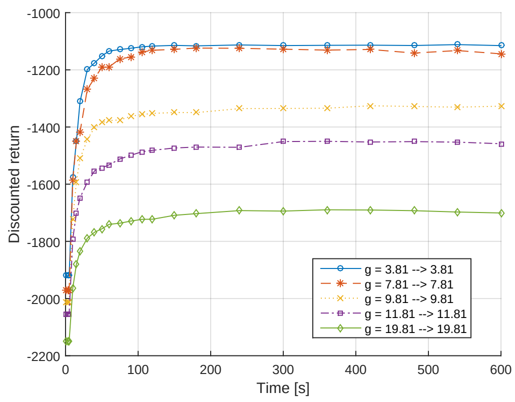


Figure 10-14: Online LSPI: Discounted return as a function of time for different gravitational acceleration, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

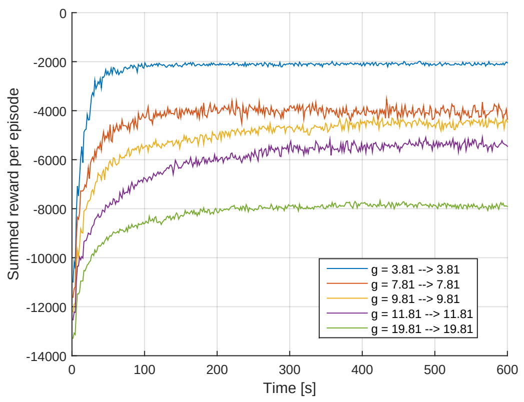


Figure 10-15: Online LSPI: Learning score as a function of time for different gravitational acceleration, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

Now we will study how the algorithm copes with changes in the conditions. In Figures 10-16 and 10-17 it can be seen that the sudden change in gravitational acceleration has a clear effect on the scores. After the change, however, it can be seen that the policy start improving, and hence the agent is adapting to the new conditions. Comparing these Figures to Figures 10-14 and 10-15 it can be seen that only the change to 19.81 m/s^2 does not reach the converging learning score that was reached with no change in conditions. Having a closer look to what this means to the behavior of the pendulum, Figures 10-18 and 10-19 that show the time history of angle and reward are interesting to study. It can be seen that when changed to the low gravitational acceleration (3.18 m/s^2), the motor becomes over time strong enough to swing up the pendulum in one go. For $g = 19.81$ m/s^2 it can be seen that in the end the motor needs a lot of swings, to reach the upright position. However, it was found that for an exploration rate ϵ of 0.2, there is too much random action, to reach the upright condition at all. For $\epsilon = 0.05$ however, the controller is able to reach the upright position.

Following up on the previous conclusion on the impact of the exploration rate on the ability to cope with changing conditions, we will focus on the change to 19.81 m/s^2 . Figures 10-20 and 10-21 show the performance for different, constant exploration rates. Looking to the left figure, it can be seen that to obtain a good policy, higher exploration rates pay off, both before and after the conditions change. This makes sense, since the higher exploration rates

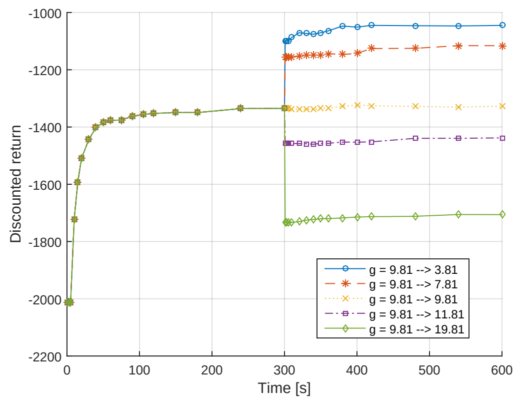


Figure 10-16: Online LSPI: Discounted return as a function of time for different changes in g at 300 s, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

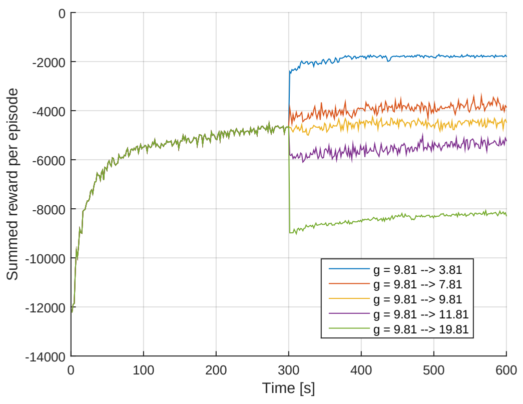


Figure 10-17: Online LSPI: Learning score as a function of time for different changes in g at 300 s, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

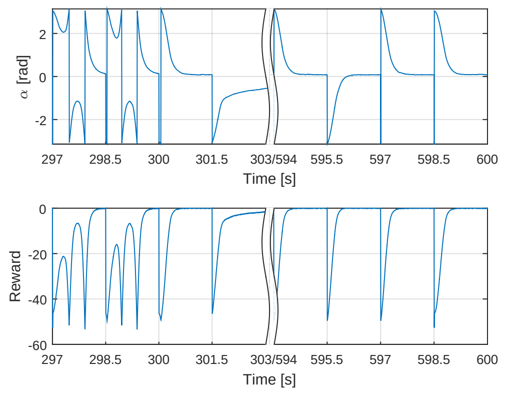


Figure 10-18: Online LSPI: Time history of the angle and reward, with a jump in g from 9.81 to 3.81 m/s^2 at 300 s, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs", note the gap in the x-axis

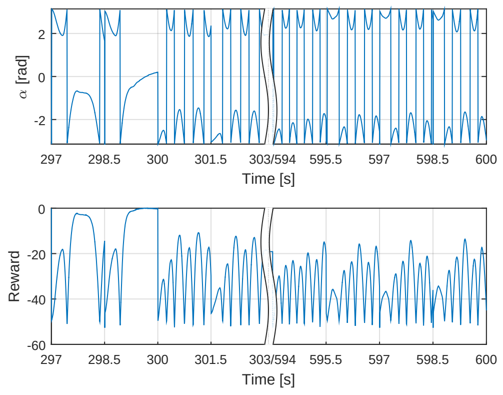


Figure 10-19: Online LSPI: Time history of the angle and reward, with a jump in g from 9.81 to 19.81 m/s^2 at 300 s, episodes last 1.5 s, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs", note the gap in the x-axis

result in a higher spread in samples, and hence in a better policy. However, looking to the right figure, it can be seen that too high ϵ results in bad online performance while learning. Since the task is hard with $g = 19.81$ m/s^2 , too high exploration yields in inability to swing up the pendulum. As such, it is found out that for this change in conditions, the moderate exploration rate of 0.05 yields best results.

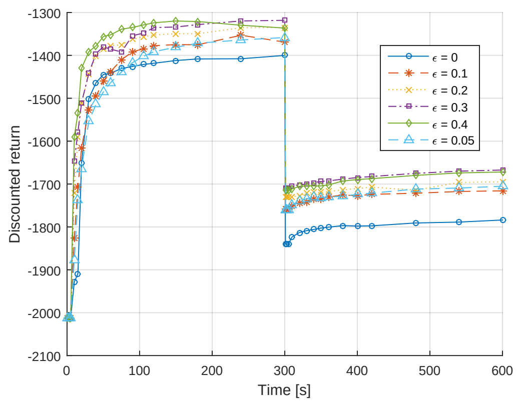


Figure 10-20: Online LSPI: Discounted return as a function of time for different exploration rates, $K_\theta = 1000$, g changes at 300 s from 9.81 m/s^2 to 19.81 m/s^2 , using "hanging down ICs"

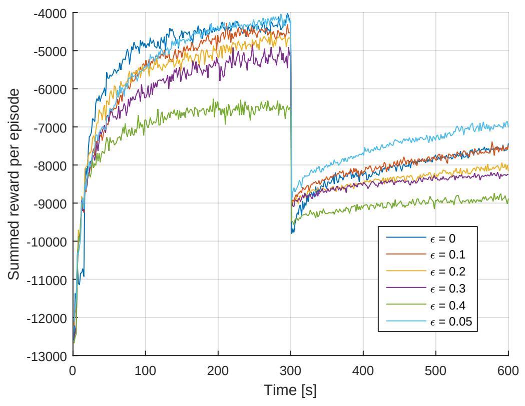


Figure 10-21: Online LSPI: Learning score as a function of time for different exploration rates, $K_\theta = 1000$, g changes at 300 s from 9.81 m/s^2 to 19.81 m/s^2 , using "hanging down ICs"

Gradual changes in conditions

In this section we will study what happens when the conditions change gradually instead of sudden, and the effect of the exploration rate on this. As for the previous section, the gravitational acceleration is changed to 19.81 m/s^2 , however, this is done gradually in a time span of 300 seconds.

In Figures 10-22 and 10-23, the results are found for this gradual change in conditions. It can be seen that the performance seems to follow the change in conditions. For all different exploration rates, the score at 600 s is close to the scores found after 600 s in Figures 10-20 and 10-21, where the sudden change in conditions was applied. In the latter figures, at 600 s the controller has had already 300 s of the same conditions. It can be concluded that when the conditions change gradually, the algorithm is able to adapt fast enough to maintain a successful policy. This conclusion is supported by Figure 10-24 which shows the time history of the angle and the reward, in the last 10 episodes (585 s - 600 s) with changing conditions. It can be seen that most of the times the agent keeps succeeding in swinging up the pendulum to zero angle, making the reward go up to the maximum, while the conditions are changing.

10-1-4 Study on forget factor

When the conditions change, the set of samples the controller uses to update the policy still largely consists of samples gathered in the old conditions. As such it is expected that when the conditions change, a forget factor could help. It makes the controller value recent samples more, and forget the older ones. Figures 10-25 and 10-26 show the performance of the algorithm with changing conditions for different forget factors. Out of this plot three interesting cases are selected: $\eta = 1$ (no forgetting), $N_{0.01} = 50000$, which corresponds to a

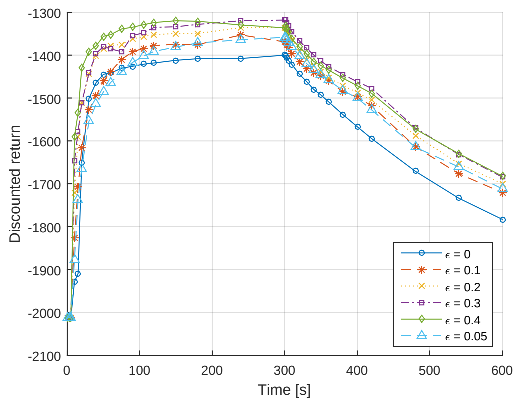


Figure 10-22: Online LSPI: Discounted return as a function of time for different sensor exploration rates, $K_\theta = 1000$, g starts changing at 300 s gradually from 9.81 m/s^2 to 19.81 m/s^2 , using "hanging down ICs"

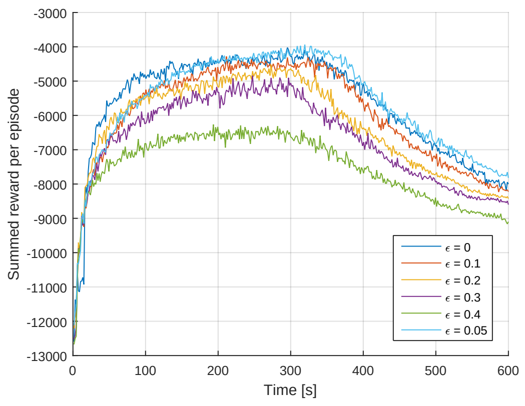


Figure 10-23: Online LSPI: Learning score as a function of time for different exploration rates, $K_\theta = 1000$, g starts changing at 300 s gradually from 9.81 m/s^2 to 19.81 m/s^2 , using "hanging down ICs"

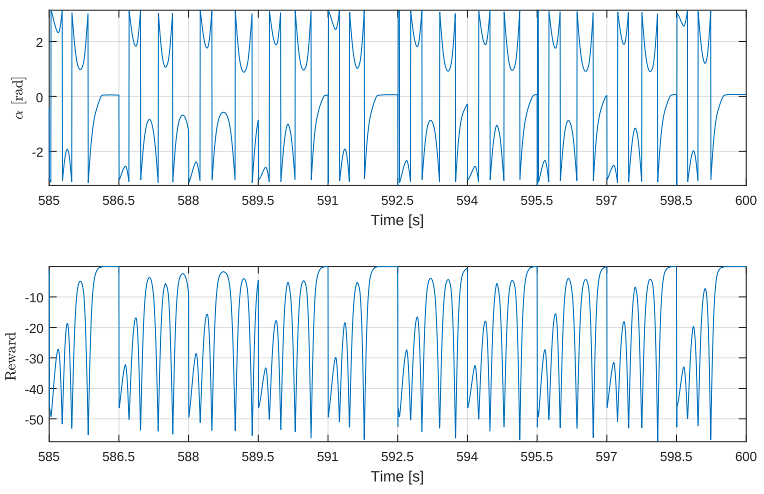


Figure 10-24: Online LSPI: Time history (last 10 episodes) of the angle and reward, in gradually changing conditions with forget factor 0.99991, $\epsilon = 0.05$, $K_\theta = 1000$, episodes last 1.5 s, using "hanging down ICs"

forget factor of 0.99991, since this shows a rather good average performance, and $N_{0.01} = 10000$, which corresponds to a forget factor of 0.9995, because that one shows poor behavior.

Figure 10-27 shows the performance for these different forget factors for online LSPI. First of all it can be concluded that a too low forget factor (i.e. much forgetting) yields poor results. This can be explained by the fact that in for example the case of the forget factor of 0.9995,

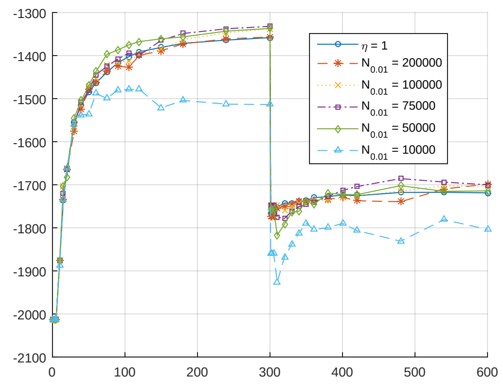


Figure 10-25: Online LSPI: Discounted return as a function of time for different forget factors, $K_\theta = 1000$, g changes at T = 300s suddenly to 19.81 m/s^2, using "hanging down ICs"

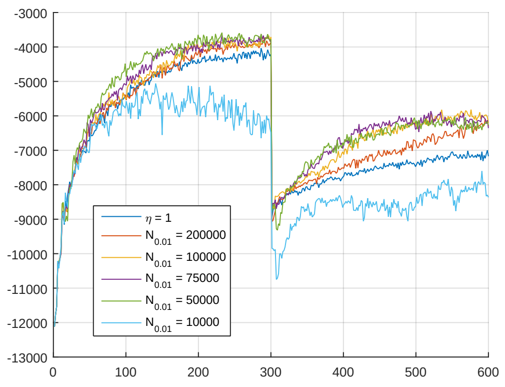


Figure 10-26: Online LSPI: Learning score as a function of time for different forget factors, $K_\theta = 1000$, g changes at T = 300s suddenly to 19.81 m/s^2, using "hanging down ICs"

only 10000 samples are used, since all samples older than 10000 time steps are practically discarded. It is found that a set of 10000 samples is not enough to uphold a good value function. Second it can be seen that a forget factor of 0.99991 yields better performance than a forget factor of 1.This can be explained by the fact that the samples at the beginning are forgotten after some time. These samples do not contribute too much, since they are all around the hanging down position. After the change in conditions, the forget factor of 0.99991 also clearly performs better. It forgets the old conditions gradually, and by that allows for a better policy in the new condition.

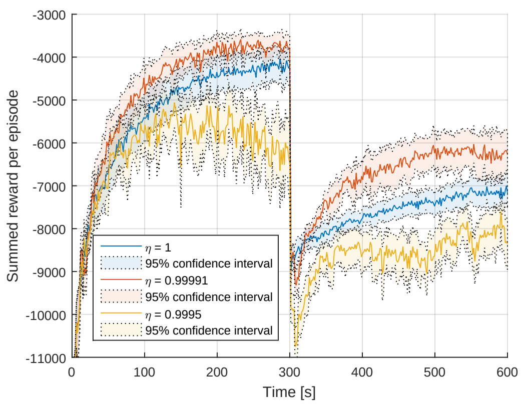


Figure 10-27: Online LSPI: Summed reward per episode as a function of time for different forget factors, $K_\theta = 1000$, $\epsilon = 0.05$, g changes at 300 s from 9.81 m/s^2 to 19.81 /ms, using "hanging down ICs"

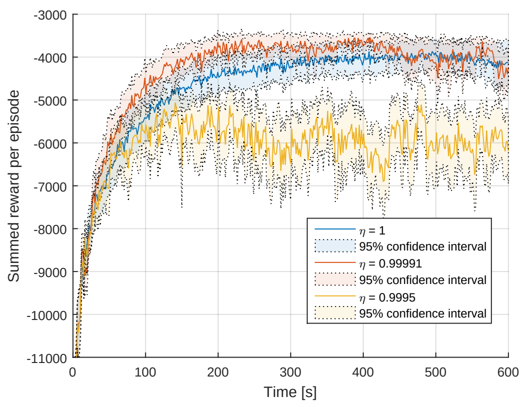


Figure 10-28: Online LSPI: Summed reward per episode as a function of time for different forget factors, $K_\theta = 1000$, $\epsilon = 0.05$, constant g, using "hanging down ICs"

In Figure 10-27 it seems that after improving the policy, at 200 s the score goes down again for $\eta = 0.9995$. To study this effect further, Figure 10-28 is plotted as well, showing the performance when the conditions do not change. Here it can be seen that for the forget factor $\eta = 0.9995$, the score fluctuates some, but stabilizes around a certain performance.

10-1-5 Results for the rotational pendulum problem with distributed initial conditions

As was mentioned in the conference paper in Part I, the initial conditions grid was changed from always starting from the hanging down position, to the following grid: $\alpha \in \{-\pi, \pi/2, 0, \pi/2\}$ rad and $\dot{\alpha} \in \{-10\pi, -3\pi, -\pi, 0, \pi, 3\pi, 10\pi\}$ rad/s. In the paper it was shown that the benefit of the human demonstration is a lot less with these distributed ICs, since with these initial conditions, a large spread in samples is obtained anyway. This phenomenon is also reflected in Figures 10-29 - 10-32. It can be seen that for the found policy, the exploration rate is a lot less critical. Different constant exploration rates and different exploration decay factors yield the same discounted return. The difference in online learning score still is evident however. This is obviously because for higher exploration rates, while learning the number of random actions is higher, and hence the score becomes lower.

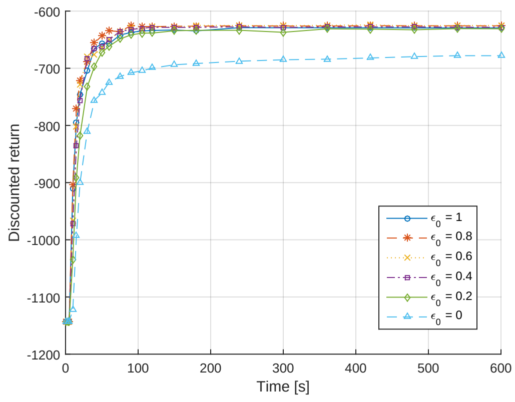


Figure 10-29: Online LSPI: Discounted return as a function of time for different constant exploration rates, $K_\theta = 1000$, $\epsilon_d = 1$, using “distributed ICs”

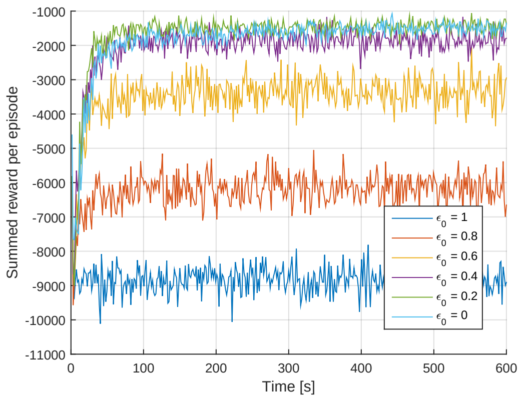


Figure 10-30: Online LSPI: Learning score as a function of time for different constant exploration rates, $K_\theta = 1000$, $\epsilon_d = 1$, using “distributed ICs”

Comparing Figures 10-31 and 10-32 (with the distributed ICs) to Figures 10-3 and 10-4 (with hanging down ICs), it is seen that for this “distributed ICs” problem the time to learn a good policy is a lot shorter than for the initial problem. It can be concluded that there is a dependence on the type of problem on the time to learn a good policy. Furthermore, it was shown in the paper (Part I) that for this easier “distributed ICs” problem, which requires less intelligent exploration, the benefit of the human demonstrator is small, compared to a fully random demonstrator.

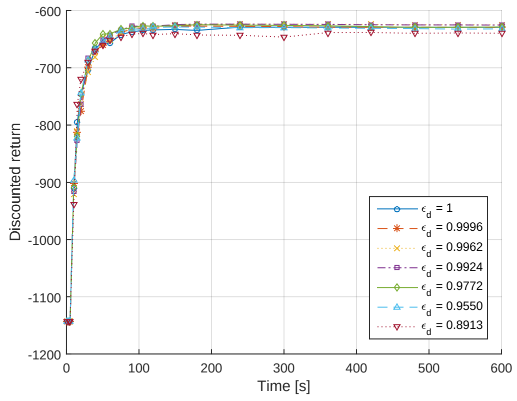


Figure 10-31: Online LSPI: Discounted return as a function of time for different exploration decay factors, $K_\theta = 1000$, $\epsilon_0 = 1$, using "distributed ICs"

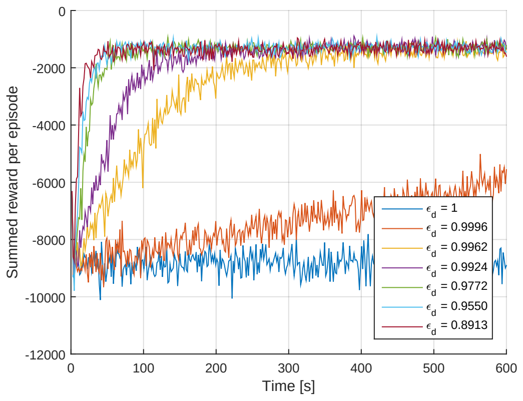


Figure 10-32: Online LSPI: Learning score as a function of time for different exploration decay factors, $K_\theta = 1000$, $\epsilon_0 = 1$, using "distributed ICs"

10-2 IOLSPI with the use of human demonstrations

This section provides some additional results on the IOLPSI algorithm, and the different human demonstration sets that were generated.

10-2-1 Explanation on different human demonstrations

For clarity Table 10-1 is repeated in this section. In addition to these five human demonstration sets, each setting (perform/explore and corruption percentage) is repeated, to see whether the results are reproducible. Figures 10-33 and 10-34 present the results of these different human demonstration sample sets within the IOLSPI algorithm. The repeated demonstrations are denoted with the same number as there counterpart, with an "a" added in the legend. From the figures it can be clearly seen that the results are similar for the repeated human demonstrations. It can be concluded that the approach used is reproducible.

Table 10-1: Overview of human demonstration sample sets.

Sample set number	Corruption	Description
1	0%	tried to perform well
2	0%	tried to explore
3	50%	tried to perform well
4	33%	tried to perform well
5	33%	tried to explore

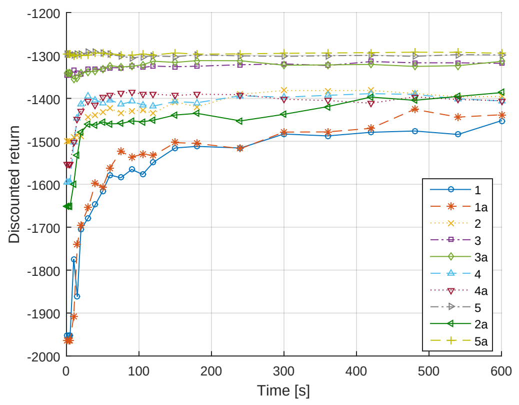


Figure 10-33: IOLSPI: Discounted return as a function of time for different human demonstration sets, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

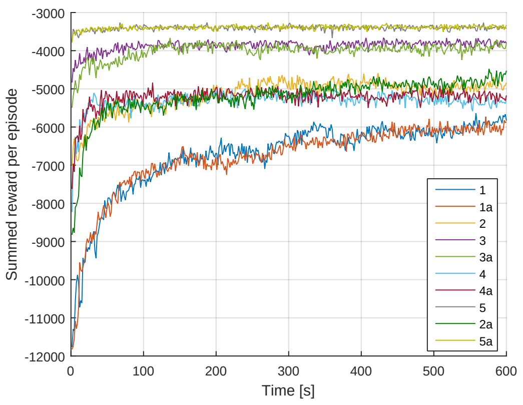


Figure 10-34: IOLSPI: Learning score as a function of time for different human demonstration sets, $K_\theta = 1000$, $\epsilon = 0.2$, using "hanging down ICs"

10-2-2 Influence of noise

The following sections provide the results of the IOLSPI algorithm when noise is added to the system.

Noise on the action

For IOLSPI with human demonstration use, the discounted return shows no big difference for different sensor noise levels (Figures 10-35 and 10-36). For the learning score a difference can be seen rather clearly on the other hand. Moderate noise is fine, but the high noise levels mix things up. The pendulum is still able to swing up, but the balancing upright is hard, since the harsh disturbances. In addition the action is also penalized in the reward function, which also results in lower scores. The reason that this difference is not so pronounced in the discounted return plot, is that the discounted return weighs the beginning of the episode more than the end. In the beginning there is some benefit because of the action, since high action values can help to swing up the pendulum faster. The most important conclusion to draw here is the same as for online LSPI: The IOLSPI algorithm is robust to moderate noise levels.

Sensor noise

Figure 10-37 clearly shows that the high sensor noise results in worse performance, since the action will be often selected based on the wrong state. For moderate noise levels, the algorithm has no problems. The same can be seen in Figure 10-38. Moderate noise levels are fine, though extreme noise levels result in problems.

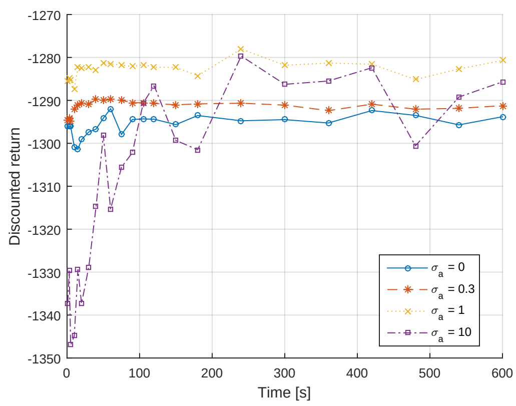


Figure 10-35: IOLSPI: Discounted return as a function of time for different action noise levels, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_a = \{0, 0.3, 1, 10\}$ V, using "hanging down ICs"

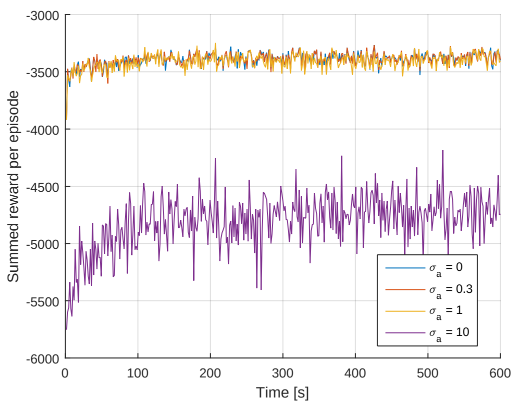


Figure 10-36: IOLSPI: Learning score as a function of time for different action noise levels, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_a = \{0, 0.3, 1, 10\}$ V, using "hanging down ICs"

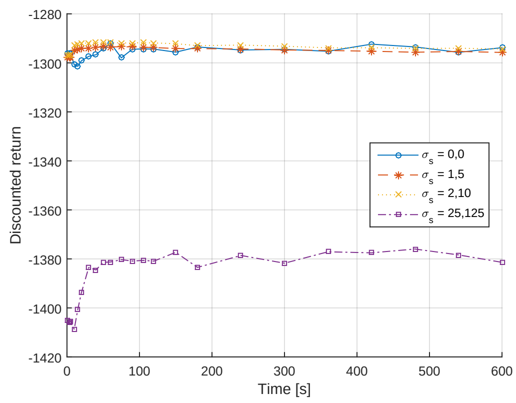


Figure 10-37: IOLSPI: Discounted return as a function of time for different sensor noise levels, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_s = \{0, 1, 2, 25\}$ deg and $\{0, 5, 10, 125\}$ deg/s, using "hanging down ICs"

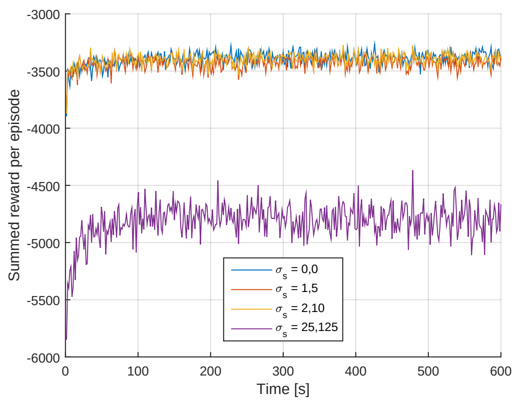


Figure 10-38: IOLSPI: Learning score as a function of time for different sensor noise levels, $K_\theta = 1000$, $\epsilon = 0.2$, $\sigma_s = \{0, 1, 2, 25\}$ deg and $\{0, 5, 10, 125\}$ deg/s, using "hanging down ICs"

10-2-3 Behavior in changing conditions

In this section we will look at the behavior in changing conditions of the IOLSPI algorithm using a human demonstration sample set. First a sudden change in conditions is studied, changing the gravitational acceleration from 9.81 m/s^2 to 19.81 m/s^2 at 300 s. After this a gradual change from 9.81 /ms to 19.81 m/s^2 is simulated.

Sudden change in conditions

Figure 10-39 shows the summed reward per episode while learning with a sudden change in conditions. The first 300 seconds are as expected: The performance is from the start on good. With higher exploration, the performance is worse, due to the higher percentage of random actions. After the change in conditions, the higher exploration rates are not too interesting to study. This is, as was mentioned in the online LSPI section of this chapter, because the task is too difficult to reach the upright position with this level of randomness in the actions. For $\epsilon = 0.1$, the agent comes close, but does not succeed in reaching a satisfactory policy. Only with no exploration, or with $\epsilon = 0.05$ a satisfactory performance is reached.

In Figure 10-40 the same results are repeated, however, with the use of a moderate forget factor of 0.99991. It can be seen that for the high exploration rates, the performance after 300 s only becomes worse, it has forgotten too much of its exploration of the state space obtained from the human demonstrations. For the low exploration rates, two interesting things can be seen: First of all the performance, just after the conditions change, is lower than without forget factor. This is attributed to the fact that much of the human demonstration, which has samples throughout a large portion of the state space, has been forgotten, which results in worse results. The second observation is that with the forget factor included, the performance goes up a lot faster than without the forget factor. This is because the new samples, including the changed conditions, are valued more than the outdated ones when the forget factor is used.

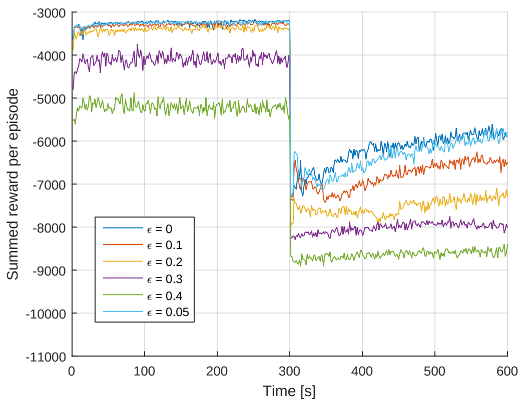


Figure 10-39: IOLSPI with HD: Learning score as a function of time for different exploration rates, $K_\theta = 1000$, $\eta = 1$, g changes at T = 300 s instantly to 19.81 m/s^2, using "hanging down ICs"

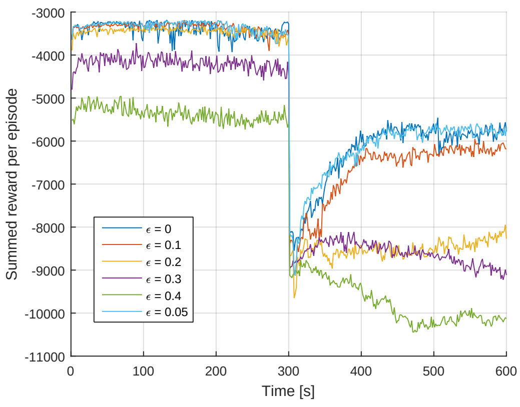


Figure 10-40: IOLSPI with HD: Learning score as a function of time for different exploration rates, $K_\theta = 1000$, $\eta = 0.99991$, g changes at T = 300 s instantly to 19.81 m/s^2, using "hanging down ICs"

Gradually changing conditions

Figures 10-41 and 10-42 show the behavior in gradually changing conditions without and with forget factor respectively. Similar results can be seen compared to the online LSPI algorithm. The performance gradually changes along with the changing conditions. Again, for too high exploration rates the problem becomes unsolvable. For the lower exploration rates it can be seen that the performance at 600 s is a little lower compared to the sudden change in conditions, where the agent has had 300 s time to adapt to the changed conditions. The differences are not that big however, so it can be concluded that for moderate changes in conditions, the algorithm is able to keep up. Studying the forget factor shows the same conclusions as for the suddenly changed conditions case: For the first 300 s the forget factor makes the performance decrease a little, and when the conditions start changing, the forget factor is beneficial. Studying Figure 10-42 more closely, an interesting phenomenon can be observed for $\epsilon = 0$. From 540 seconds onward, the performance stabilizes. This is because at 540 seconds, the algorithm has found a policy which allows swinging up the pendulum in three swings, whereas in the first 300 seconds, two swings were enough. It was seen that having found this policy at 540 seconds, the gravitational acceleration change is small enough from that moment on, to keep being able to swing up the pendulum in three swings and thus the performance stays constant.

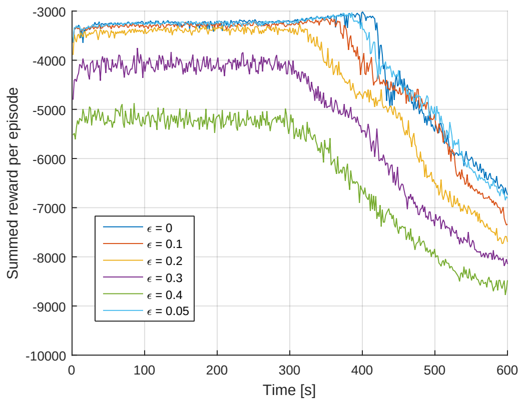


Figure 10-41: IOLSPI with HD: Learning score as a function of time for different exploration rates, $K_\theta = 1000$, g changes at T = 300s gradually to 19.81 m/s^2, $\eta = 1$, using "hanging down ICs"

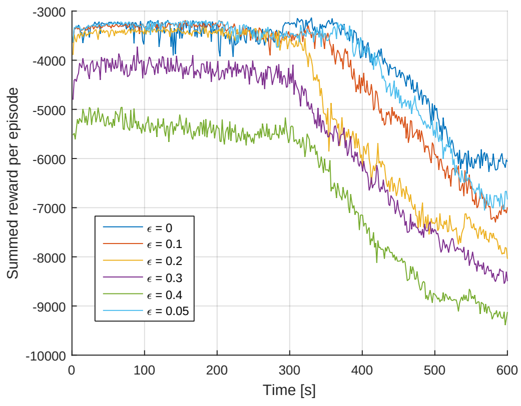


Figure 10-42: IOLSPI with HD: Learning score as a function of time for different exploration rates, $K_\theta = 1000$, g changes at T = 300s gradually to 19.81 m/s^2, $\eta = 0.99991$, using "hanging down ICs"

Chapter 11

Quadrotor Slung Load Problem

This chapter gives an additional description of the quadrotor slung load problem that was introduced in section IV of Part I. Furthermore, some additional observations on this problem implementation are presented.

11-1 Addition to the problem statement

In addition to the parameters mentioned in the conference paper in Part I, for completeness some more information on the implementation of the slung load problem is presented. In this problem an $\epsilon-$greedy policy is used. For the IOLSPI, which already has information on the problem, a constant exploration rate of 0.01 is used. For the online LSPI algorithm, which has no prior knowledge, this constant exploration rate results in poor results. To alleviate this problem, an exponentially decaying exploration rate is used for the online LSPI algorithm. The exploration rate starts at 1, and decays to 0.01 in the end. With this decaying exploration rate, enough samples are gathered randomly to eventually reach a satisfactory policy.

For the use of the radial basis functions, a lower and upper limit for the state variables is needed, since between these limits the RBF centers are spaced. Table 11-1 shows these limits, which are found by empirically studying the boundaries of the state space.

Table 11-1: Limits of state variables

State variable	Lower limit	Upper limit
x [m]	-6	0
$\dot{x}$ [m/s]	-4	4
θ [rad]	-$\pi/2$	$\pi/2$
$\dot{\theta}$ [rad/s]	-π	π

Furthermore, the following should be noted concerning the implemented target reaching logic. The agent is rewarded when it drops the load in the basket. In order to do this, it has to swing up the load high enough to reach the basket. It would have been logical to penalize it when the load hits the side of the basket. However, since this basket side is rather small, it would ask for a fine RBF grid to specifically differentiate between the states in which the side of the basket is hit and in which the load just flies over the edge. As a consequence, this penalty is not included. This means that the load cannot crash in the side of the basket, but will just go through as if there was no side of the basket. However, it still has to swing up the load, since it only gets rewarded when it drops the goal in the basket from above. As such, it still is an interesting problem to solve with this reinforcement controller.

11-2 Observations on the slung load problem

Several observation can be made, when studying the quadrotor slung load problem as described in this thesis:

- The initial conditions (ICs) used are designed rather artificially. The quadrotor starts at an arbitrary position, with an arbitrary slung angle. In reality it would make more sense to always start at a location away from the basket, whereas here the quadrotor in some episodes starts already above the basket. This choice of initial conditions results in a high variety in the samples anyways, similar to what we saw with the “distributed ICs” for the rotational pendulum problem. When the initial conditions are narrowed down to a smaller region, between x = -5 m and and -4 m, we see that the online LSPI algorithm is not able to converge towards a satisfactory policy in a reasonable number of iterations. The IOLSPI algorithm does not have this problem, since it already has a sample set with a spread over the whole domain.
- The number of RBFs is definitely the limiting factor for this problem. Adding more RBFs, over all dimensions, would probably result in better performance. It was found however, that when we use more RBFs, more samples are needed to fit a proper value function. As such, adding more RBFs adds computation time to the problem in two ways: It slows down the solving of the Least Squares Policy Iteration, and it requires the algorithm to run for more episodes.

Chapter 12

Conclusion

In this thesis we have shown that for some systems, human demonstrations can improve the performance of reinforcement learning in terms of learning time and safety.

The following research sub-questions are formulated for this research:

1. How can human demonstrations be used within Reinforcement Learning (RL) algorithms?
2. How do RL algorithms using human expert demonstrations compare to each other, and how do they compare to existing techniques in terms of safety and learning time?
3. In what way should the human demonstrate in order to maximize the benefit of the demonstration for the RL controller?
4. How does the effect of the human demonstration depend on the specific controlled system?

In the literature review (Chapter 4), the first question is partly answered. It is found that in literature two ways of using human demonstrations within RL for quadrotor can be identified. The first one is using a set of human demonstrations to identify a model of the system, which is used offline to learn a policy. This method is model based and offline, and hence not easily extensible to more complex vehicles, and not able to adapt to changing conditions, which both are demands formulated in this research. The second way of using human demonstrations, is to use them within a least squares policy iteration algorithm, in order to have a good initial guess of a satisfactory policy. This approach can be used online and does not require a model. As such this approach fulfills the formulated demands for the algorithm, and hence is the approach studied in this research.

In this research, as is described in Part I, this online, model-free approach found in literature is extended by combining two state of the art RL algorithms. Offline Least Squares Policy Iteration (LSPI) and online LSPI are combined to an algorithm named Informed Online Least

Squares Policy Iteration (IOLSPI). This algorithm is able to use human demonstration sample sets to improve performance in terms of learning time and safety. The IOLSPI algorithm uses the information present in a batch of samples, which can be obtained by a human demonstration, to kick start the online LSPI algorithm, which continues to learn online.

It was shown that the IOLSPI algorithm fulfills the demands for an autonomous controller: It is fully model-free and it can adapt to changing conditions. To improve the adaptability of the algorithm, a forget factor was implemented, to value outdated samples less than the more recently gathered ones, to focus on the current situation. It is found that a mild forgetting behavior results in better performance in changing conditions. With a mild forget factor present, the IOLPSI algorithm needs over 40000 time steps less to obtain a similar performance after the conditions have changed, compared to the algorithm without the use of a forget factor.

The second question is partly answered in the preliminary simulation chapter. A simple, discrete, 2D simulation shows that the use of human demonstrations within a Q-learning algorithm proves to be successful. Less time to learn is required to reach a satisfactory policy.

In Part I and Part III the second question is answered for continuous state problems. The example studied is the rotational pendulum problem, where a mass has to be swung up from a hanging down position. Here it was found that using human demonstrations, a good policy is found straight away, whereas non-initialized online LSPI needs 60000 time steps (300 s) to converge. In addition the final policy found using human demonstrations was 10% better in terms of cumulative reward than the non-initialized online LSPI.

With a second studied system the benefit in terms of safety of the IOLSPI algorithm was shown. In this quadrotor slung load drop off task it was shown that the number of crashes while learning is reduced by 10% to 50% using the IOLSPI algorithm, compared to a non-initialized reinforcement learning algorithm.

The third and fourth research questions were answered together in Part I. It was found that the benefit of the use of human demonstrations is largest when the human uses its understanding of the problem to efficiently explore the state space, rather than only giving perfect demonstrations of the task at hand. As a result, the benefit of using human demonstration in combination with IOLSPI is larger for problems in which some understanding of the problem is needed to reach a larger portion of the state space. This is the case in the aforementioned swing up pendulum problem. The rotational pendulum problem was changed in such a way that each episode of the problem was started at a random state, and as such no understanding of the problem is needed to explore the whole state space. For this problem it was found that the IOLSPI found a good policy straight away, but in this case the non-initialized online LSPI algorithm found already after 20000 time steps (100s) a good policy, which was not different from the policy found by the IOLSPI algorithm using human demonstrations.

With the second case study, a quadrotor slung load drop off task, the limitation of the algorithm became apparent. A uniformly distributed grid of radial basis functions was used to approximate the Q-function. With a growing number of state parameters, the number of basis functions grows exponentially. For higher dimensional problems, the computational effort becomes too high for present day computers.

For future research it would be interesting to combine the IOLSPI algorithm with methods that choose the radial basis functions in an intelligent way, or use other ways of defining basis functions. When the number of basis functions is limited, the computational effort stays low. With this addition in place, it would be interesting to apply the algorithm also to multiple actuator systems, with continuous actions. With these extensions, the algorithm could be applied to a 3D quadrotor simulation, and eventually on a real quadrotor. This will be a next step towards model-free and adaptive control supporting the autonomy of these complex vehicles.

Appendix A

Additional Results of the Preliminary Simulation

In this appendix the influence of the initialization of the parameters in the 2D simulation are presented. The reference setting is the following: randomness $\epsilon = 0.01$, learning rate α= 0.01, initialization of the Q-matrix $Q_0 = -50$. The effect of variations in these parameters are discussed in the following sections.

Effect of Q-initialization

The value with which the Q-values are initialized has a high influence on the speed of convergence of the algorithm and, therefore, on the average return over 2000 iterations. Figure A-1 shows the return for each iteration and the average return over all passed iterations for the reference setting. It can be seen that using human demonstrations the convergence is faster and hence the average return as well. Figure A-2 shows the average return as a function of different initializations of the Q-values (Q_0). It can be seen that for both approaches there is a benefit of taking a negative value for Q_0. Initializing it at zero, will mean in the case for Q-learning that the algorithm will be optimistic in the face of uncertainty, i.e., all state-action combinations that are not visited yet, have a value of zero, which is in any case higher than all visited state-action combination since these are negative because of the reward structure. It has been seen that only 2000 iterations are not enough to make it converge towards a satisfactory result. For the human demonstration algorithm, a Q_0 of zero will work counterproductive: This means that all state-action combinations that are visited within the human demonstration runs, are deemed worse, since they will have a negative value, than all non-visited states.

It can be seen from the figure that a value of -50 yields good results for both algorithms.

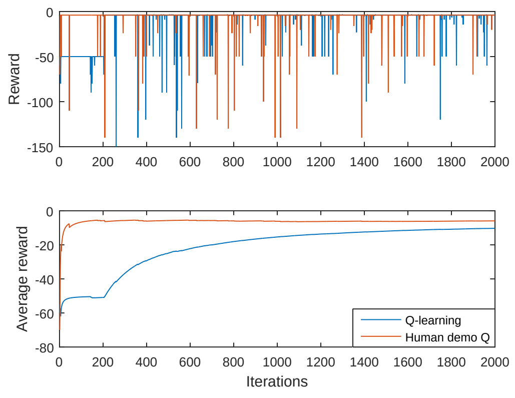


Figure A-1: Return and average return, $\epsilon = 0.01$, $\alpha = 0.01$, $Q_0 = -50$

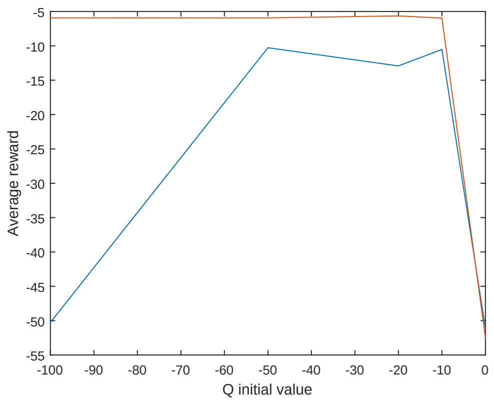


Figure A-2: Average return as a function of Q_0, $\epsilon = 0.01$, $\alpha = 0.01$

Effect of greediness

Figure A-4 shows the impact of the greediness of the algorithm. It can be seen that for the human demonstrations algorithm the lower the exploration rate, the higher the average reward is. This is because the initial policy is good, and exploration does not yield in finding a better policy within these 2000 iterations. This means that the higher the randomness, the more bad results will occur. This lowers the average return. Figure A-3 shows this for an exploration rate of 0.001. It can be seen that since the initial policy is already good, the average is going to be high and constant.

For the Q-learning algorithm the influence is different. With too low exploration, the good policy is not found, and hence the average return will end up low. For high exploration rates, however, the influence of nonsense random iterations will start deteriorating the average too much. As a result, it can be seen that an exploration rate of between 0.001 and 0.05 is optimal for this problem.

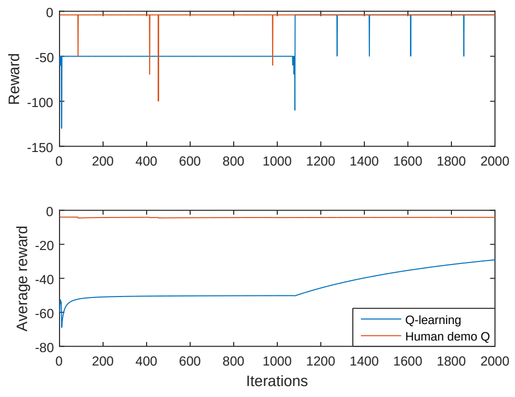


Figure A-3: Return and average return, $\epsilon = 0.001$, $\alpha = 0.01$, $Q_0 = -50$

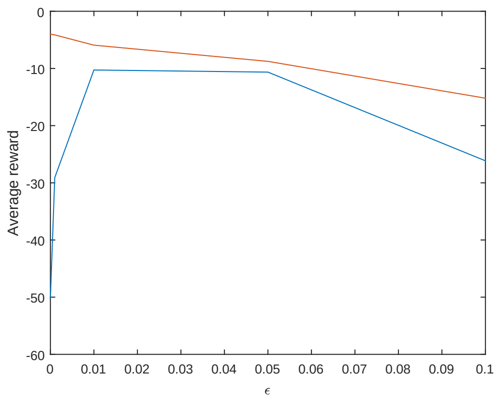


Figure A-4: Average return as a function of ϵ, $\alpha = 0.01$, $Q_0 = -50$

Effect of learning rate

The learning rate α is a measure on how aggressive the values are updated after a state has been visited compared to its previous value. As can be seen in Figure A-5 and A-6 when the learning rate is low but non-zero, it does not make a big difference. Too high learning rates, however, give too aggressive behavior, and hence are not desirable.

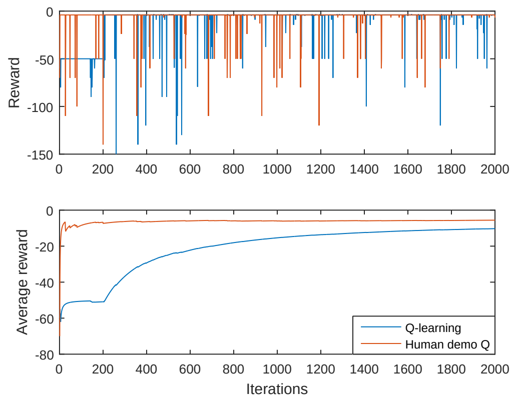


Figure A-5: $\epsilon = 0.01$, $\alpha = 0.001$, $Q_0 = -50$

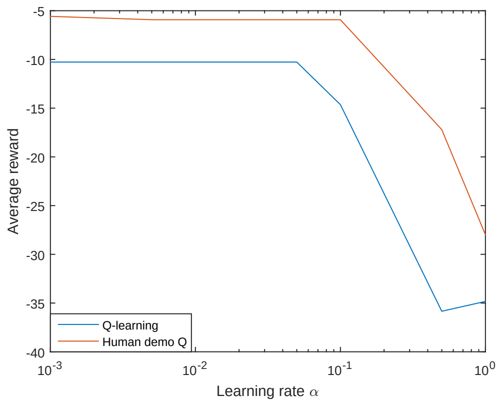


Figure A-6: Average return as a function of α, $\epsilon = 0.01$, $Q_0 = -50$

Bibliography

[1] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1279–1291, 2012.

[2] G. C. H. E. de Croon, M. A. Groen, C. De Wagter, B. Remes, R. Ruijsink, and B. W. van Oudheusden, "Design, aerodynamics and autonomy of the delfly," *Bioinspiration & Biomimetics*, vol. 7, no. 2, 2012.

[3] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Learning swing-free trajectories for uavs with a suspended load," in *IEEE International Conference on Robotics and Automation, 2013*, pp. 4902–4909, 2013.

[4] A. G. Barto and R. S. Sutton, *Reinforcement learning: An introduction.* Cambridge, Massachusetts: The MIT Press, 1998.

[5] L. Busoniu, B. De Schutter, R. Babuska, and D. Ernst, "Exploiting policy knowledge in online least-squares policy iteration: An empirical study," *Automation, Computers, Applied Mathematics*, vol. 4, pp. 521–529, 2010.

[6] M. Vaandrager, R. Babuska, L. Busoniu, and G. Lopes, "Imitation learning with non-parametric regression," in *IEEE International Conference on on Automation, Quality and Testing, Robotics, 2012*, pp. 91–96, 2012.

[7] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[8] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information Processing Systems*, vol. 19, 2007.

[9] M. Wiering and M. V. Otterlo, *Reinforcement Learning State-of-the-Art.* Springer, 2012.

[10] I. Palunko, A. Faust, P. Cruz, L. Tapia, and R. Fierro, "A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots," in *IEEE International Conference on Robotics and Automation, 2013*, pp. 4896–4901, 2013.

[11] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*. Boca Raton, Florida: CRC Press, 2010.

[12] R. Figueroa, A. Faust, P. Cruz, L. Tapia, and R. Fierro, "Reinforcement learning for balancing a flying inverted pendulum," in *Intelligent Control and Automation*, pp. 1787–1793, 2014.

[13] H. Bou-Ammar, H. Voos, and W. Ertel, "Controller design for quadrotor UAVs using reinforcement learning," in *Proceedings of the IEEE International Conference on Control Applications*, pp. 2130–2135, 2010.

[14] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Automated aerial suspended cargo delivery through reinforcement learning," *Artificial Intelligence*, vol. 1, pp. 1–18, 2014.

[15] M. Shaker, M. N. R. Smith, S. Yue, and T. Duckett, "Vision-based landing of a simulated unmanned aerial vehicle with fast reinforcement learning," in *International Conference on Emerging Security Technologies*, pp. 183–188, 2010.

[16] I. Palunko, P. Donner, M. Buss, and S. Hirche, "Cooperative suspended object manipulation using reinforcement learning and energy-based control," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 885–861, 2014.

[17] A. R. B. Whillas, "Sound synthesis techniques." `http://alexander.whillas.com/Sound/SuperCollider/Sound+Synthesis+Techniques`, 2015.

[18] C. C. de Visser, "Introduction lecture AE4320 System Identification of Aerospace Vehicles," 2015.

[19] I. Palunko, R. Fierro, and P. Cruz, "Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2691–2697, 2012.

[20] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, "Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning," in *International Conference on Intelligent Robots and Systems*, pp. 468–473, 2005.

[21] M. G. Lagoudakis and R. Parr, "Model-free least squares policy iteration," in *Proceedings of Neural Information Processing Systems*, vol. 2, pp. 1547–1554, 2001.

[22] M. G. Lagoudakis, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[23] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska, "Online least-squares policy iteration for reinforcement learning control," in *American Control Conference , 2010*, pp. 486–491, 2010.

[24] A. Faust, P. Ruymgaart, M. Salman, R. Fierro, and L. Tapia, “Continuous action reinforcement learning for control-affine systems with unknown dynamics,” tech. rep., Department of Computer Science, University of New Mexico, 2013.

[25] S. R. B. Santos, S. N. J. Givigi, and C. L. N. Júnior, “An experimental validation of reinforcement learning applied to the position control of UAVs,” in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 2796–2802, 2012.

[26] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[27] C. Poppe, “Aggressive quadrotor maneuvering by using nonlinear dynamic inversion with pseudo-control hedging and flight mode transition.” Master of Science thesis at TU Delft, 2014.

[28] H. O. Wang, K. Tanaka, and M. F. Griffin, “An approach to fuzzy control of nonlinear systems: Stability and design issues,” in *IEEE Transactions on Fuzzy Systems, 1996*, vol. 4, pp. 14–23, 1996.

[29] R. A. Jarvis, “On the identification of the convex hull of a finite set of points in the plane,” *Information Processing Letters*, vol. 2, pp. 18–21, 1973.