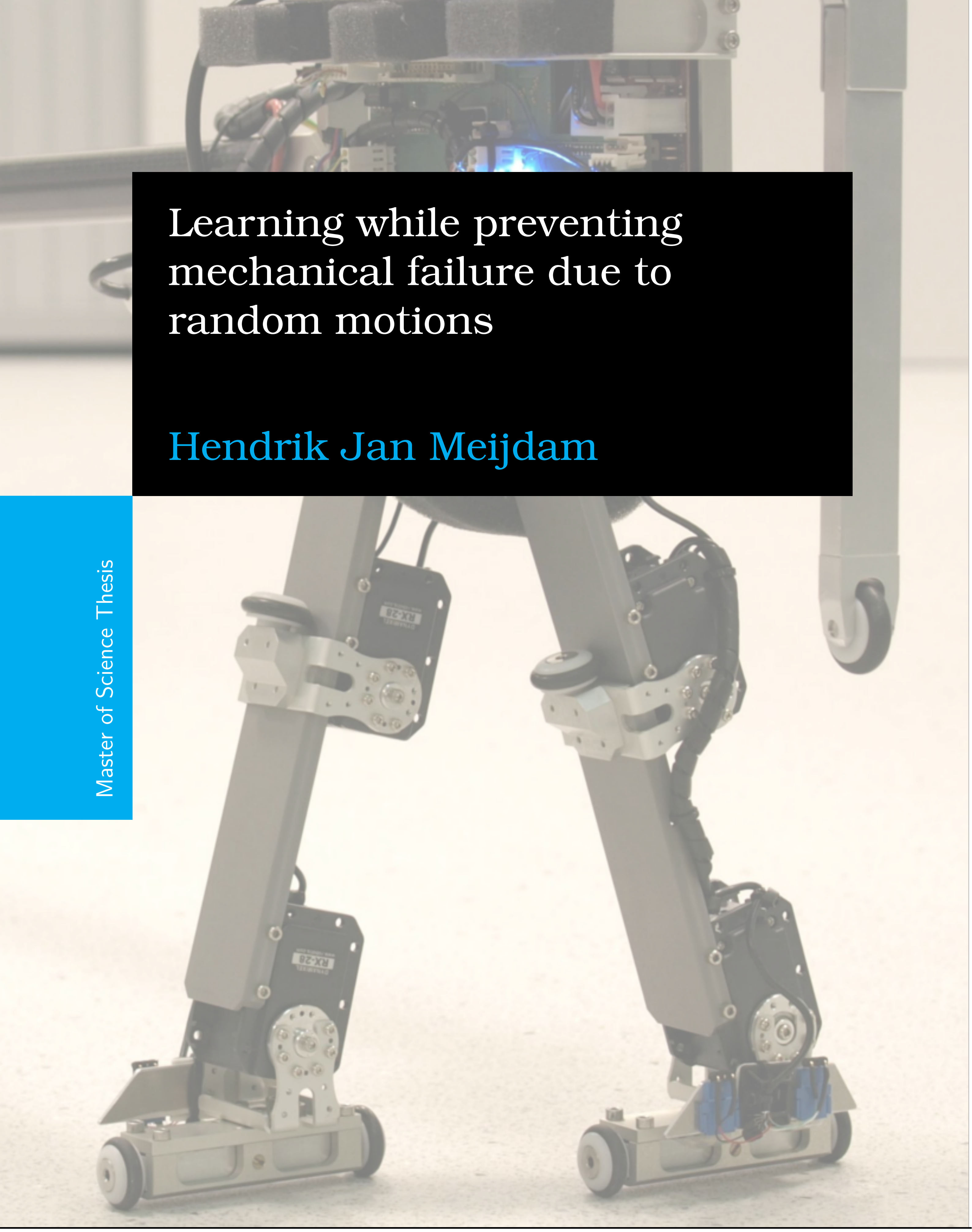


Learning while preventing mechanical failure due to random motions

Hendrik Jan Meijdam

Master of Science Thesis

Delft University of Technology

Faculty of Mechanical, Maritime and Materials Engineering

Learning while preventing mechanical failure due to random motions

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

Hendrik Jan Meijdam

Tuesday 28th May, 2013

Faculty of Mechanical, Maritime and Materials Engineering · Delft University of Technology

Delft University of Technology
Department of
BioMechanical Design

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

Learning while preventing mechanical failure due to random motions

by

Hendrik Jan Meijdam

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: Tuesday 28th May, 2013

Supervisor(s):	Dr. Wouter Caarls
Reader(s):	Prof. Pieter Jonker
	Prof. Robert Babuska
	Dr. Heike Vallery

Abstract

In this thesis one of the negative effects of learning from scratch on the durability of LEO is analysed. LEO is one of the bipedal walking robots of the TU Delft Robotics Institute. It uses Reinforcement learning to learn a stable and energy efficient walking gait. LEO's learning algorithm causes its gears to fail faster during the initial learning phase than during the optimisation phase. One of the reasons for the low mean time between failure (MTBF) is the learning algorithm itself. The learning algorithm initially causes random motions which in turn cause high stresses in the gears and mechanical failure. The MTBF due to these motions can be predicted. This MTBF can be increased by adapting the learning algorithm in various ways. We investigated 5 algorithms that increase the MTBF and compared them to SARSA(λ) learning. In general, increasing the MTBF decreases the learning performance. Three of the investigated algorithms are unable to increase the MTBF while keeping their learning performance approximately equal to SARSA(λ). Two algorithms are able to do this: the PADA algorithm and the low-pass filter algorithm. In case of LEO, the MTBF can be increased by a factor of 108 compared to SARSA(λ) learning. This indicates that in some cases, failures due to random motions can be prevented without decreasing the learning performance.

Table of Contents

List of Figures

List of Tables

Acknowledgments

There are a number of persons I would like to thank for their support during my masters project. First of all I would like to thank my supervisor Wouter Caarls. He always had an open mind when I came up with new ideas and when I tried to explain these ideas in my thesis his feedback greatly improved its quality. I also would like to thank him for his support when I was writing a paper for the IROS convention.

Secondly I would like to thank Michiel Plooij who also supported me during the writing of this paper. I also applied his comments on how to increase the quality of this paper on my thesis.

Thirdly I would like to thank Martijn Wisse, Daniël Karssen, Sander van Weperen, Wouter Wolfslag, Guus Liqui Lung, Tim Vercruyssen and other members of the Delft biorobotics lab who created a motivating atmosphere during lunches and tolerated my loud and annoying experiments.

Lastly I would like to thank my family, friends and girlfriend for their love and support.

Delft,
Tuesday 14th May, 2013

Chapter 1

Introduction

Learning motions from scratch on physical systems causes mechanical stresses in these systems. They are especially high in the initial learning phase, in which random motions frequently occur. The stresses can cause failure before a motion has been learned and there are three main causes for them. The first cause is that the robot needs to perform a task and this task can only be completed by stressing some components. The second is that harmful states need to be visited before the system learns to prevent them. These two causes cannot be prevented in the initial learning phase because they are an essential part of the learning process. The last cause is the behaviour of the robot while learning from scratch. Without having had any feedback the robot starts by performing high frequency random motions, which cause stresses in the robot. These motions are not an essential part of learning therefore it should be possible to reduce them without negatively effecting the learning process. The problem statement of this thesis is therefore formulated as:

Can the mean time between failure of robotic systems be increased by decreasing random motions due to learning, without negatively effecting the learning process?

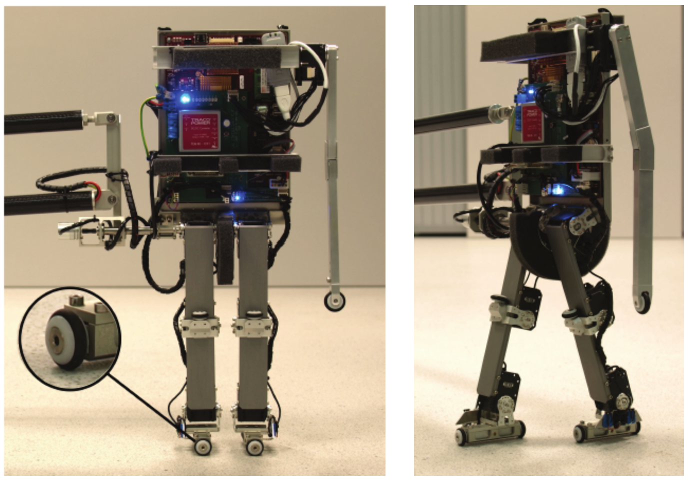
The focus of this thesis will be on one robotic system in particular. It will be on the robot LEO. The mean time between failure of LEO is small when it tries to learn motions from scratch. In the next chapters the properties of LEO will be covered. After that the problem is analysed with an inverted pendulum simulation, experiment and finally a simulation of LEO itself. In appendix B the content of this thesis is summarised in a paper.

Chapter 2

Learning to walk

LEO is displayed in figure 2-1. LEO uses Reinforcement learning to learn itself a policy for stable and optimal walking. First the general principle of Reinforcement learning will be covered in this chapter. Then the properties of the Markov decision process and several learning methods will be covered. After that the design and the shortcomings of LEO will be treated.

2-1 Reinforcement learning



(a) Front view **(b)** 3D view

Figure 2-1: LEO, one of the bipedal robots of the Delft Biorobotics Laboratory viewed from different angles [1].

Reinforcement learning is one of the most active areas of machine learning. In [2] reinforcement learning is defined as systems who learn themselves to accomplish something they want.

2-1-1 General principle

The learning part of the system is the agent and the rest is called the environment. The agent interacts with the environment by performing an action and getting a reward and a state returned by the environment. This process is illustrated in figure 2-2. The possible states for the agent to be in is defined by the set S and the possible actions the agent is allowed to take are defined by the set A. Reinforcement learning is a semi-supervised learning method. Semi-supervised implies that there is no supervisor that defines what action the agent is supposed to take. Instead it gets a hint in the form of a reward. The policy of an agent determines which action it takes for each state and is indicated with the π symbol. The agent tries to learn which action to take in each state to get the maximum rewards. In other words it tries to learn an optimal policy. In order to find this policy it needs to take actions which have not been taken before. This is called exploration. The speed at which the policy converges to the optimal policy is the learning speed. The interaction between the agent and the environment is divided into time steps. These time steps breakdown into episodes, which are natural sub-sequences of steps. The state of the environment at the end of the episode is the terminal state. A trial is a sequence of episodes and a learning process is a combination of these trials.

There are many types of environments. It can be a separate piece of software like a simulator or it can be a real physical environment like a robot. Reinforcement learning with a real physical robot as environment is called on-line reinforcement learning. The episodes of these processes are called on-line episodes.

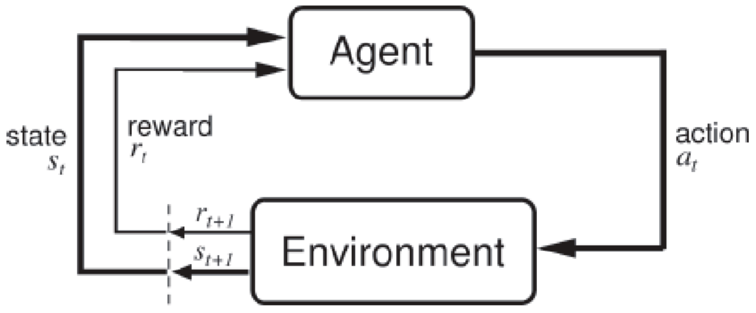


Figure 2-2: An illustration of how the agent and environment interact is given in [2].

2-1-2 Markov decision processes

It is convenient for a Reinforcement learning processes to be a Markov decision process. For Markov decision processes the convergence to an optimal policy can be guaranteed. A Markov decision process is a process which satisfies the Markov property. The Markov property is satisfied if the state succeeds in retaining all information of the process needed to obtain the probability of reaching a future state. So if the Markov property is satisfied, then there is a probability for each state to reach a certain other state, given that a certain action is taken. This probability is defined by:

$$P^a_{ss'} = Pr\left\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\right\} \tag{2-1}$$

All these probabilities combined are the transition function, P. The expected reward can be obtained by:

$$R^a_{ss'} = E\left\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\right\} \tag{2-2}$$

In these equations s, a and s' are the state, the action and the future state respectively. All these expected rewards combined are the reward function, R. A Markov decision process is defined by the tuple <S, A, P, R>. In which the S is the set of states, A the set of actions, P the state transition probabilities and R the set of rewards.

With these equation 2-1 and 2-2 the state value can be computed. This value depends on the expected reward in the future and can be calculated with equation 2-3. All state values combined are the state-space.

$$V^\pi(s) = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} = \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \tag{2-3}$$

The γ parameter defines the horizon for which the future reward is computed. These state values can be used to learn the optimal policy. If the state values are optimal then the policy is optimal. In order to speed up the process of learning the optimal policy, state-action value can be calculated. This value depends on the expected reward in the future after taking a certain action in a certain state. It can be calculated with equation 2-4. All state-action values combined are the state-action space.

$$Q^\pi(s,a) = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} = \sum_{s'}\mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \tag{2-4}$$

With these state-action values the policy can be improved by selecting the action, in each state, that has the highest value. The equation to obtain this policy is given in 2-5.

$$\pi'(s) = arg\ \max_a Q^\pi(s,a) \tag{2-5}$$

A policy that always complies with 2-5 is called a greedy policy. The new policy, π', is at least as good as the previous policy. At the end of the learning process the policy will not change and the state-action values will not increase. At this point the policy is optimal .

Optimistic initialisation

One way of increasing the learning speed is by optimistically initiating the state-action values [2]. The learning process is initiated by setting the state-action values higher than their actual value at that time. This will encourage the agent to take actions it has not taken before. If the state-action values are given a random and optimistic value the agent starts the learning process by taking random actions. For instance, the agent is in a certain state, s, and performs a certain action a_i which has the highest value. It will lower the state-action value $Q(s, a_i)$. Now assuming that the agent stays in state s the agent will probably perform an other action a_j at the next time step. The value of this state-action will also be lowered and will probably not be performed the next time step. This process continues until the state-action values are close to their actual value.

2-1-3 SARSA(λ)

The optimal policy can be approximated by several methods. One of them is SARSA(λ). SARSA is a Temporal difference prediction method. This method uses bootstrapping to approximate the state-action values. The idea behind bootstrapping is guessing based on a guess. Equation 2-3 can be rewritten as:

$$V^\pi(s) = E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \right\} \tag{2-6}$$

So the state values can be updated with the reward and future state value. This is known as bootstrapping and can also be done for the state-action values. The state-action values can be updated without knowing all the future rewards and are updated with every time step. The previous state-action values are moved with a constant step-size parameter, α. They are updated with equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2-7}$$

From this equation it is clear to see that the previous state-action is rewarded a part of the next state-action value. To reward also state-actions which where visited before the previous state-action an eligibility trace can be used. An eligibility trace speeds up convergence at the cost of increased computational demand. This is caused by the increased load on the CPU and RAM memory. The eligibility of every state is stored and updated with the following algorithm:

$$e_t(s,a) = \left\{ \begin{array}{c} 1 \text{ if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) \text{ if } s \neq s_t \text{ and } a \neq a_t \end{array} \right\} \text{for all } a \text{ and } s$$

The λ value is the trace-decay parameter and defines the length of the eligibility trace. All the state-action values are updated according to their eligibility. State-actions which were visited recently have a high eligibility. The difference between the previous value of the state-action and the new, the temporal difference error, is calculated with equation:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{2-8}$$

This value is used to update all state-action values with the following algorithm:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta e_t(s,a) \text{ for all } a \text{ and } s$$

2-1-4 Actor-Critic

The SARSA(λ) method uses one structure to approximate the state-action values. From these values the optimal policy is derived. The optimal policy can also be derived explicitly by separating this structure in a state value structure and a policy structure. This is called an actor-critic method. Actor-critic methods are also temporal difference prediction methods like SARSA. The policy structure is the actor and the state value structure is the critic. During the learning process the actor will select an action, a_t, based on the state. Using this action the state will change and the critic can deliver some criticism. The temporal difference error is given by:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2-9}$$

This δ_t value represents the difference between the expected state value and the experienced state value. It is not used to update state-action values as with SARSA but this δ_t is used to criticize the actor. If it is positive the actor should be adapted to select a_t more often in that state. If it is negative the actor should be adapted to select that action less often in that state.

The actor does not evaluate all possible actions before it chooses one therefore actor-critic methods can work with continuous action spaces. Another advantage is that they can learn explicitly stochastic policies because they can learn the optimal probabilities of selecting different actions.

2-1-5 Function approximation

The size of the state or state-action value structures cause problems with the amount of memory needed to store them. Also the learning speed reduces with increasingly big structures. Generalisation can be used to decrease these problems. Generalization can be used to make an approximation of the value functions. For instance, the state values, V_t, are not represented by a table but by a function which is depending on a vector, $\overrightarrow{\theta_t}$. The amount of $\overrightarrow{\theta_t}$ components is less than the state values therefore a range of states get new values if they change. The $\overrightarrow{\theta_t}$ components can be tuned to represent the state value function by applying a gradient descent method. A gradient descent method adjusts the parameter vector, after each example, in the direction that would reduce the error on that example the most. For state values an example of this is:

$$\overrightarrow{\theta_{t+1}} = \overrightarrow{\theta_t} + \alpha \left[V^\pi - V_t(s_t)\right] \nabla_{\overrightarrow{\theta_t}} V_t(s_t) \tag{2-10}$$

This can also be done for state-action values. The method guarantees convergence to a local optimum. Here V^π is the true value of the state and $\nabla_{\overrightarrow{\theta_t}} V_t(s_t)$ is the gradient of the state value with respect to the vector. The true value of the state is not available in practice. Using an unbiased estimate of the true state values also guarantees convergence to a local optimum.

Calculating the gradient of the state value with respect to the vector, $\overrightarrow{\theta_t}$, can be difficult. To prevent this problem a special case of function approximation can be used. If a linear function of the parameter vector, $\overrightarrow{\theta_t}$, is used then the gradient can be calculated easily. The linear value function:

$$V_t(s) = \overrightarrow{\theta_t}^T \overrightarrow{\phi_s} \tag{2-11}$$

Has the following gradient:

$$\nabla_{\overrightarrow{\theta_t}} V_t(s_t) = \overrightarrow{\phi_s} \tag{2-12}$$

An added advantage of a linear function is that it has only one optimum. This means that if the method is guaranteed to converge to a local optimum it also converges to the global optimum. For linear function approximation, in combination with Temporal difference learning, convergence has been proven.

One often applied form of linear function approximation is tile coding. The state is exhaustively partitioned into a number of overlapping tiles. If a state is in a tile the corresponding

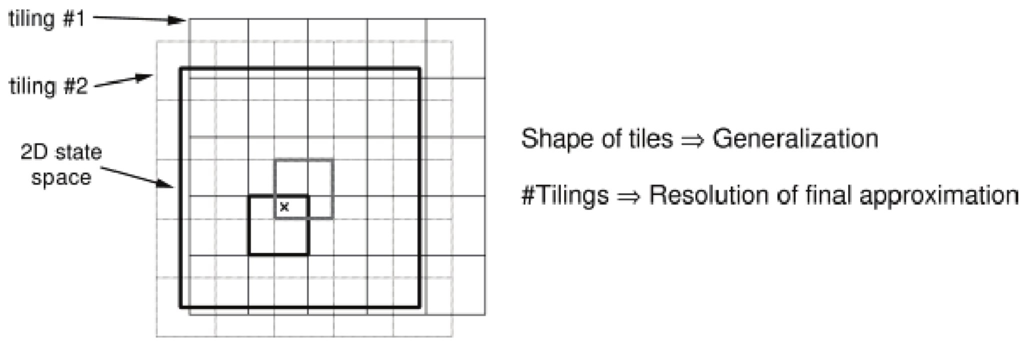

tiling #1
tiling #2
2D state space
Shape of tiles ⇒ Generalization
#Tilings ⇒ Resolution of final approximation

Figure 2-3: An example given in [2] of a two-dimensional state partitioned into tiles.

$\overrightarrow{\theta_t}$ element has a value of one and else it has a value of zero. An example of a two-dimensional state partitioned into tiles is given in figure 2-3.

Function approximation speeds up convergence but the resulting policy might become sub-optimal if the function approximation has too few parameters or if the function is shaped inappropriately.

2-2 Policy gradient

During the learning process a sequence of actions can produce a high frequency signal. These signals can damage physical systems. A smooth action signal during learning can be ensured by parametrising the policy. The dimension of the learning problem can be reduced by this parametrisation because the policy can be approximated by less parameters than the state-action values. However, in order to accurately approximate the policy, while using a small number of parameters, knowledge of the learning problem is required. Therefore, a policy needs to be parametrised specifically for its learning problem. This limits the ability of the policy to adapt to unexpected situations.

Policy search methods tune the policy parameters to maximise the rewards given. One form of policy search is the policy gradient method. A local optimal policy can be obtained by looking at the effect of parameter changes on the reward. The expected return of a policy depends only on its parameters. It is calculated by:

$$J(\pi_\theta) = E\left\{\sum_{k=0}^{H} \gamma^k r_k\right\} \tag{2-13}$$

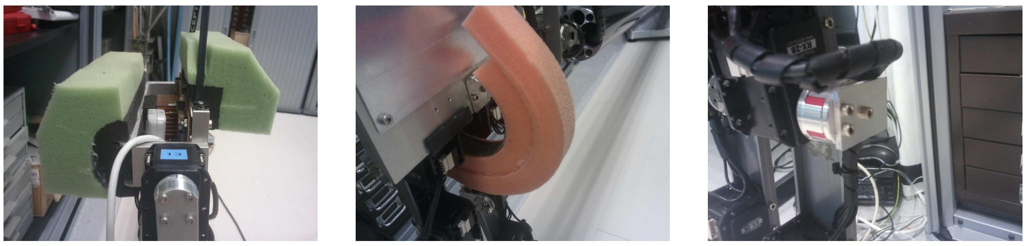
Here γ is between 1 and 0 and H is the horizon of the learning problem. If the episode has a terminal state then the expected return of the policy can be obtained. A simple case is if the γ value is 1 and the horizon is over the entire episode. In this case all rewards over the entire episode are summed. If the expected return is known then the parameters of the policy can be tuned. The policy parameters are tuned with the following equation:

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J(\pi_\theta)|_{\theta=\theta_k} \tag{2-14}$$

In this equation α_k is the learning rate. The gradient of the parameters with respect to the expected return, $\nabla_\theta J(\pi_\theta)|_{\theta=\theta_k}$, is difficult to compute analytically. Therefore, in some cases this gradient is approximated. For instance, the gradient can be approximated by looking at the expected return of three policies. This is done in [5]. These policies have some parameters that differs between policies. By looking at the performance of all three policies the gradient can be approximated. For instance, if decreasing the parameter value reduces the expected return and increasing this value increases the return then the parameter value is increased. Some policy gradient methods adjust the policy parameters based on more functions than J alone. Policy gradient methods and policy search methods in general limit the policy which makes it hard to apply them without having prior knowledge of the learning problem and therefore it is not applied to LEO.

2-3 LEO's design

2-3-1 Design requirements



(a) Upper-body safety pads **(b)** Lower-body safety pads **(c)** elastic element

Figure 2-4: The different compliant parts that reduce impact forces on LEO.

The objective of LEO is to find a successful walking gait without having any prior knowledge. LEO's walking gait is designed to be a Markov decision process and it's episodes end when it falls. It is one of the first robots that tries to learn with On-line Reinforcement Learning. It uses tile-coding and a SARSA(λ) algorithm for on-line and on-policy learning. Its state-action values are randomly and optimistically initialised. Other robots which have used this learning method restricted the control policy, had an initial policy and state-action values learned by simulation or first learned a policy by mimicking examples. LEO is designed to be able to learn without prior knowledge and therefore LEO has been designed to fulfil ambitious requirements. In [3] it is stated that in real robots the Markov property of the environment is violated. LEO however is designed to keep the violation of this property to a minimum. The requirements are stated in [1] as follows:

1. "The robot can walk over a period of days and is robust against falls and self-collisions."
2. "The robot can observe state s, which holds all information relevant to the learning problem."
3. "The effect of action a in every state s is predictable."
4. "The sampling time is constant."

5. "T must be static within a time frame of tens of hours."

6. "The time between measurement s_k and action a_k is zero."

7. "The number of degrees of freedom should allow for currently viable learning tasks, as well as more complex ones."

2-3-2 Design

LEO is designed to comply with these requirements however LEO cannot comply with two of them. LEO meets requirement two, three, four, six and seven but cannot meet requirement one and five.

Degrees of freedom

To comply with requirement seven three simplifications are applied to LEO's walking. The first simplification is that the motion is two dimensional. LEO is connected to a boom which restricts it to a plane. This boom is visible in figure 2-1 at the left side of the robot. The boom rotates around one axis which is perpendicular to the floor. The second is that the legs of LEO have three joints; an ankle, knee and hip joint. These joints have only one axis of rotation. The third simplification is that the torques of the hip joints and the swing leg knee joint together define the total motion of LEO. LEO has one arm with which it can stand up before beginning the walking motion but this arm is not actuated during learning. The angles of its leg joints together with the angle of its upper body define its state therefore the robot has seven degrees of freedom.

Action effect

LEO learns to control the three motors therefore, to comply with requirement three, the action space has three dimensions. The hip joint motors and motor of the swing leg knee are controlled. The actions correspond to a voltage put on the motor of a joint. The motors delivering these torques are Dynamixel RX-28 motors. The effect of temperature change on these motors is compensated via additional voltage.

State observation

All joint angles and velocities of both legs and the upper body are measured to comply with requirement two. The Dynamixel RX-28 motors are coupled to their joints with gearboxes and have potentiometers which measure the angle and velocity of them. The potentiometers have an angle range of $[10, 290]^\circ$ with a minimum accuracy of $\pm 4^\circ$. With these measurements the state of the robot can be calculated. LEO has pressure sensors in its feet which detect whether its feet are flat on the ground or not. The angle between the boom and the hips of the robot is also measured.

Sampling

To comply to requirement four LEO uses real-time computing. It uses Linux with the Xenomai extension. A sampling time of 6666.6 [μs] has an average standard deviation of 230 [μs].

Control delay

LEO has been designed to minimize control delay to comply with requirement six. This is minimized by processing all signals on-board. Requesting position data and sending actuation commands takes a few milliseconds.

State transition

LEO cannot comply with requirement five because the system is not time-invariant. The state transition function is not time-invariant. This violates the Markov property. If two different data sets are used to predict the other, then the accuracy of these predictions decrease as the time between these sets increase.

Robustness

LEO does not comply with requirement one because, despite of different safety measures, Its parts fail after a period of minutes during learning. LEO has safety pads on his lower and upper body. These pads protect LEO's body from large impact forces from falling and can be seen in figures 2-4a and 2-4b. LEO's hip and knee joint gears are protected from large impact torques by the elastic elements displayed in figure 2-4c. These elastic elements are compressed if the motor turns without the leg turning or vice versa. In figure 2-1a one foot of LEO is magnified. There the rubber bands on which it stands are visible. These bands protect the robot from large impact forces during walking. Another set of these bands protect LEO from large forces due to self-collision when LEO stretches its legs. LEO is able to walk a limit cycle for more than 8 hours straight if it uses a preprogrammed controller. However, the mean time between failure (MTBF) drops significantly if LEO tries to learn this. The MTBF is half an hour during the optimisation phase. This can be seen in figure 2-5. If LEO tries to learn from scratch the MTBF decreases to 5 minutes. This is concluded in [3]. Weak parts of LEO are the potentiometers of the motors, the gearboxes and the pressure sensors in the feet. This is concluded in [1] and in [6].

2-3-3 Learning with Dynamixel RX-28 motors

Dynamixel RX-28 protection and Markov property violation

It is difficult to mechanically protect the Dynamixel RX-28 motors without increasing the violation of the Markov property. In order to increase the MTBF of LEO during learning its gearboxes are protected by elastic elements. The function of the elastic elements is to protect the motors by allowing displacement between the motor and the limb. This reduces the stresses that occur in the gearboxes during impacts.

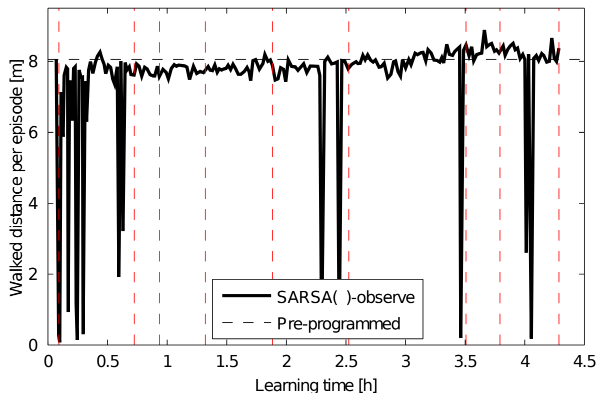
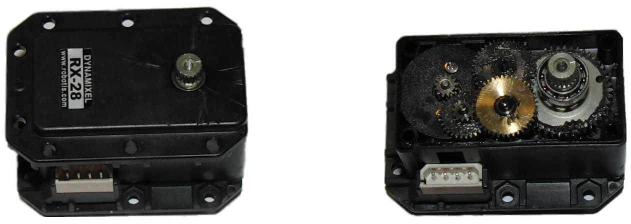


Figure 2-5: Results from an experiment in [3] in which LEO tries to optimize a policy. The vertical dashed lines represent moments on which mechanical failure occurs.

LEO's limbs are all powered by Dynamixel RX-28 motors which are controlled at 30 $[Hz]$. One of these motors is displayed in figure 2-6. In this figure the gearbox of this motor is also



(a) Closed **(b)** Opened

Figure 2-6: The Dynamixel RX-28 motors that drive LEO's limbs. On the left it is displayed with its full casing. On the right the gearbox of the motor is visible.

displayed. Documentation of this motor can be found in appendix A. The knee coupling with the elastic element is visible in figure 2-7. A schematic of this mechanical system is displayed in figure 2-8. Here I_1 is the reflected motor inertia, I_2 is the limb inertia, T_m is the torque

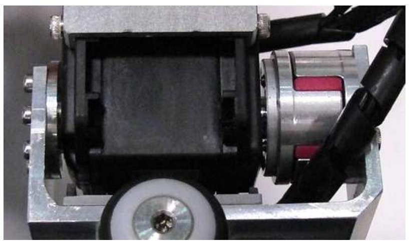


Figure 2-7: The Dynamixel RX-28 motors are coupled to LEO's limbs via elastic elements [3].

exerted by the motor, k is the stiffness of, and T_e the torque in, the elastic element. The state of LEO is defined by measuring θ_1 and $\dot{\theta}_1$ of each joint. A block scheme can be made based on the mechanics of the motor. This scheme is visible in figure 2-9. From this block scheme it is clear that each motor adds four state dimensions to the learning problem while only two are measured. By not measuring all states the Markov property is violated. To keep this violation to a minimum it is important that the system can be described by a second order

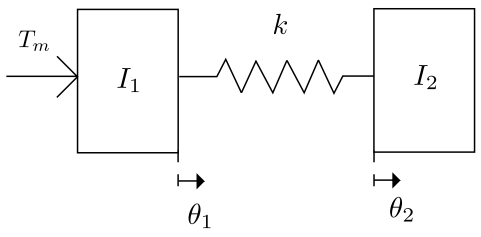


Figure 2-8: The gearbox is represented by a reflected motor inertia. The backlash in the gearbox together with the elastic element connecting the limb and motor is represented as a linear torsion spring. In this figure rotations are represented as translations.

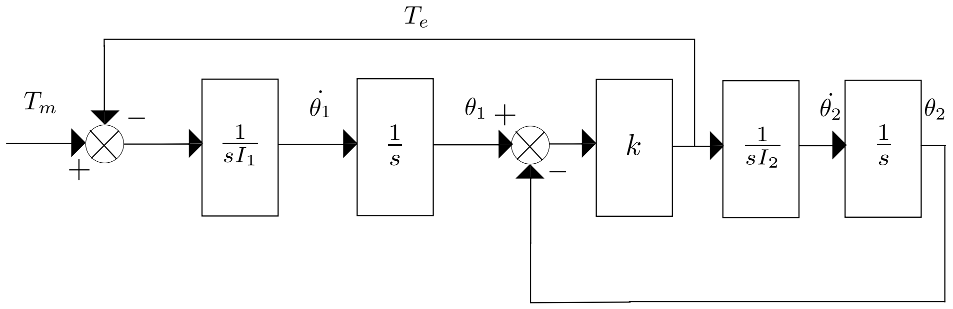


Figure 2-9: The block scheme based on the mechanics of the Dynamixel RX-28.

system accurately. The transfer function from T_m to θ_2 is:

$$\frac{T_m(s)}{\theta_2(s)} = \frac{k}{s^4 \cdot I_1 \cdot I_2 + s^2 k \cdot (I_1 + I_2)}$$

The order of the transfer function can be decreased by assuming a rigid connection. In this case a rigid connection implies that the system can be approximated by one inertia. By taking the limit from k to infinity this inertia can be calculated:

$$\frac{T_m(s)}{\theta_2(s)} = \lim_{k \to \infty} \frac{k}{s^4 \cdot I_1 \cdot I_2 + s^2 k \cdot (I_1 + I_2)} = \frac{1}{s^2(I_1 + I_2)} \tag{2-15}$$

An infinite stiffness corresponds to zero displacement between the reflected motor and the limb inertia. Therefore, the state of the limb does not have to be measured. Infinitely stiff mechanical components do not exist in practice. A rigid connection can still be assumed if the poles of the transfer function are much faster than the control frequency. The transfer function from equation 2-15 has four poles which are:

$$p_{1,2} = 0$$

$$p_{2,4} = \pm\sqrt{\frac{k \cdot (I_1 + I_2)}{I_1 \cdot I_2}} i$$

The limb inertias of the robot vary and therefore each limb has its own poles. The parameters used for the knee joint are displayed in table 2-1. As a rule of thumb, A rigid connection should not be assumed when the complex pole pair is less than 5 times faster than the control frequency. In case of a knee joint, the poles are located at $\pm 442i$ while the control frequency

R_m	8.3 $[\Omega]$	K_{emf}	$1.2 \cdot 10^{-4}$ $[V \cdot s/rad]$	I_2	0.003 $[kg \cdot m^2]$
L_m	0.206 $[mH]$	G	193	k	300 $[Nm/rad]$
K_m	10.7 $[mNm/A]$	I_1	0.0032 $[kg \cdot m^2]$		

Table 2-1: The values used for both schematics.

is 30 $[Hz]$. Therefore, the pole pair is approximately 2.5 times faster. In case of the hip joints the poles are almost equal to the control frequency. Therefore, the low stiffnesses of the elastic elements increases the violation of the Markov property. The low stiffnesses are used because a higher MTBF has a higher priority than the violation of the Markov property, Another reason for the low stiffnesses are the backlash in the gearboxes and the maximum displacement of the elements compared to the minimum accuracy of the motor position. The backlash in the gearbox is approximately 0.01 $[rad]$ [3]. The maximum displacement of the elastic element of the hip joint 0.021 $[rad]$ and 0.018 $[rad]$ for the knee joint. These displacements are small compared to the minimum accuracy of the position sensor which is 0.07 $[rad]$. The increased violation of the Markov property in favour of a higher MTBF illustrates the difficult problem of keeping a small number of state dimensions while learning a complex motion on a mechanically robust robot.

Agent actions on Dynamixel RX-28 motors

The Dynamixel RX-28 does not support current control. Therefore, it is not the torque T_m generated by the motor which is the action of the agent but the voltage V_{agent} on the motors which are the agent actions. Fortunately this does not increase the violation of the Markov property. The torque is generated by a gearbox and a number of electronic elements. A schematic of these electronic elements are visible in figure 2-10. Here V_{agent} is the agent

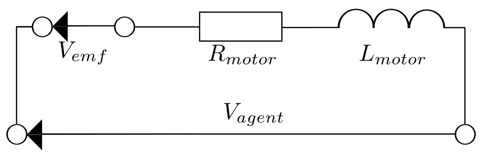


Figure 2-10: A schematic model of the electronics inside the Dynamixel. The motor can be represented by a resistance and an inductor. This inductor creates a motion opposing voltage.

action, R_m is the resistance and L_m the inductance of the motor. V_{emf} is a motion opposing voltage. This motion opposing voltage is defined by:

$$V_{emf} = K_{emf} \cdot \dot{\theta}_1 \cdot G$$

Here K_{emf} is a gain and G is the gear-ratio of the Dynamixel motor. This motion opposing voltage is subtracted from the agent voltage. The transfer function from the combined voltage to the current in the motor is:

$$\frac{i_m(s)}{V_{agent}(s) - V_{emf}(s)} = \frac{1}{R_m + L_m \cdot s}$$

And finally the torque on the reflected motor inertia is given by:

$$T_m = G \cdot K_m \cdot i_m$$

A block scheme can be created based on the electronics and the mechanical block scheme. This block scheme is visualised in figure 2-11. The dotted line separates the mechanics from the electronics. Although this block scheme suggests that the torque T_m or the current i_m needs

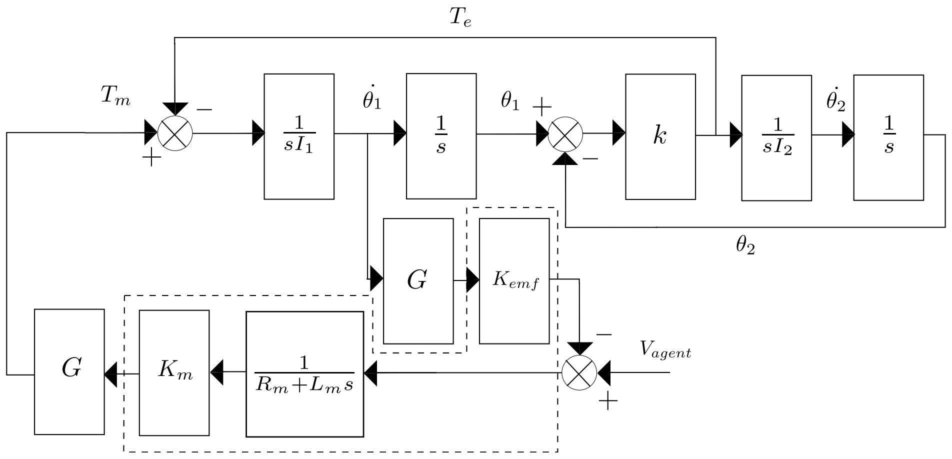


Figure 2-11: A block-scheme of the mechanics and electronics. The dotted line separates the electronical from the mechanical scheme.

to be in the state to prevent further violation of the Markov property, the state dimension does not need to be increased when the agent actions are voltages. The pole in the transfer function of the electronics is very fast compared to the control frequency. It has a pole at -R_m/L_m or -4029. Therefore, the voltage supplied to the motor, $V_{agent} - V_{emf}$, can be directly coupled to the torque on the reflected inertia. The steady state gain of the transfer function is:

$$\frac{T_m}{V_{agent} - V_{emf}} = \lim_{s \to 0} \frac{GK_m}{R_m + L_m s} = \frac{GK_m}{R} \tag{2-16}$$

Also the motion opposing voltage depends only on the speed of the motor, which is in the state of the agent. Therefore the agent can learn which voltage produces the optimal torque in each state.

2-3-4 Compensating for design issues by adapting the learning algorithm

A short MTBF prevents LEO from learning a stable walking gait while learning from scratch therefore it is necessary to increase this MTBF. In the previous section the problem of mechanically increasing the MTBF of LEO without increasing the number of state dimensions is illustrated. The MTBF of LEO can also be increased by adapting the learning algorithm. There are three main causes for the short MTBF. The first cause is that LEO needs to walk and this task can only be completed by stressing some components. For instance, the foot impact, which causes fast changes in the $\dot{\theta}_2$ signal, cannot be avoided because foot touchdown is an essential part of the walking motion. The second is that harmful states need to be visited before it learns to avoid them. For instance, in [3] LEO needs to fall approximately 5000 times in order to learn a stable walking gait. Falling also causes fast changes in the $\dot{\theta}_2$ signal. These stresses are hard to prevent because they are an essential part of the learning process in the initial learning phase. The last cause is the behaviour of the robot while learning from scratch. Due to the optimistic initialisation the agent of LEO starts by taking random actions. The random actions cause high frequency random motions and fast changes of the V_{agent} signal, which in turn cause high stresses. The random motions are not an essential part of the learning process therefore it should be possible to reduce them, by adapting the learning algorithm, without negatively effecting the learning performance.

In the next chapter the MTBF of the last gear in the Dynamixel motor will be predicted given a certain T_m signal.

Chapter 3

Mechanical failure

We need to predict the mean time between failure (MTBF) of LEO, given a certain action signal, in order to successfully adapt the learning algorithm. In this chapter we will first predict the MTBF of an action signal. Then we will do an experiment to see how the predicted MTBF and the actual MTBF relate.

3-1 Predicting the MTBF

Mechanical failure of LEO is usually caused by a gearbox breaking down. The gear that is directly attached to the output shaft is mainly responsible for mechanical failure. It is made of anodized aluminium and has 45 teeth. The material of this gear is able to withstand a number of stress cycles before it fails due to fatigue. A method to link complex combinations of cycles to fatigue failure is described by [7]. In this section the following procedure is used to predict the MTBF. First we determine the stress cycles that contribute most to fatigue failure. Second we approximate the maximum torque on the last gear during these cycles. Third we calculate the maximum stress in the last gear due to this torque. Fourth we approximate the number of times this stress cycle can be withstood. Finally we predict the MTBF based on the predicted failure.

3-1-1 Stress cycles

The condition in which cycles occur is defined by looking at the conditions in which the gearboxes actually fail. LEO is able to walk for more than 8 hours without failing while walking with a pre-programmed controller. However, the gearboxes of LEO fail after 5 minutes while performing random motion due to learning. Apart from falling, the impact of which is minimized by foam padding, the difference between the two situations is the number of backlash re-engagements (i.e. when the torque on the gear changes sign, the backlash is experienced and the gears re-engage). This leads to the assumption that the stress cycles during backlash re-engagement have the largest contribution to failure. We also assume that one backlash re-engagement leads to one stress cycle. The stress cycle can be expressed as a function of the torque while the gears re-engage.

3-1-2 Maximum torque

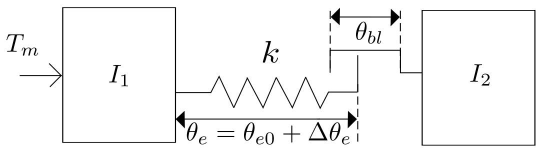


Figure 3-1: A model of the backlash re-engagement assuming no friction. In this figure the rotations are represented as linear displacements. $\Delta\theta_e$ is the angular displacement in the elastic element, θ_{bl} is the backlash that is experienced, T_m is the torque at which the gears re-engage, I_2 is the inertia of a limb and I_1 is the reflected inertia of the motor.

The gearbox can be represented by one element; an inertia which represents the reflected inertia of the motor on the joint. The total model consists of this motor with an inertial load attached to it via an elastic element (see figure 2-7). This inertial load represents a limb. A schematic of this model can be seen in figure 3-1. The second inertia represents the limb. θ_1 represents the position of the reflected inertia and θ_2 the limb position. The motor torque acts on the first inertia. The non-linear torsion spring represents the elastic element and backlash of the gearbox which transports torques between the motor and the limb. The spring is a linear torsion spring. In this model there is no energy loss due to friction in the gearbox. The dynamics can be described by a state-space model:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{k}{I_1} & 0 & -\frac{k}{I_1} & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{k}{I_2} & 0 & \frac{k}{I_2} & 0 \end{bmatrix} \cdot x(t) + \begin{bmatrix} 0 \\ \frac{T_m}{I_1} \\ 0 \\ 0 \end{bmatrix} \cdot u(t)$$

The torsion in the spring defines the torsion acting on last gear of the gearbox. The torsion in the spring can be calculated by choosing the following C matrix:

$$T_m(t) = y(t) = \begin{bmatrix} k & 0 & -k & 0 \end{bmatrix} \cdot x(t)$$

Due to backlash the coupling between the motor and the gear is lost for a short rotation of the motor. This occurs between $-\theta_{bl} < \theta_1 - \theta_2$ and $\theta_1 - \theta_2 > 0$. During this period the equations of motion change into:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot x(t) + \begin{bmatrix} 0 \\ \frac{T_m}{I_1} \\ 0 \\ 0 \end{bmatrix} \cdot u(t)$$

Assuming that the input is constant during the time that the gears re-engage the evolution of the states of this system can be calculated.

$$x(t) = e^{A\cdot t} \cdot x(t_0) + \int_0^t e^{A\cdot\tau} \cdot B \, d\tau = e^{A\cdot t} \cdot x(t_0) + \int_0^t \begin{bmatrix} 1 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \frac{T_m}{I_1} \\ 0 \\ 0 \end{bmatrix} d\tau =$$

$$e^{A\cdot t}\cdot x(t_0)+\int_0^t \begin{bmatrix} \frac{T_m}{I_1}\cdot t \\ \frac{T_m}{I_1} \\ 0 \\ 0 \end{bmatrix} d\tau$$

The states belonging to the limb inertia are not effected by the step input. The states evolve with:

$$x(t_{bl}) = e^{A\cdot t}\cdot x(t_0)+\begin{bmatrix} \frac{T_m}{2\cdot I_1}\cdot t_{bl}^2 \\ \frac{T_m}{I_1}\cdot t_{bl} \\ 0 \\ 0 \end{bmatrix}$$

before the evolution of states can be calculated an initial condition is needed. Both angles and velocities of the limb and the reflected inertia are approximately equal. The model has an initial condition from which the backlash is fully experienced:

$$x_0 = \begin{bmatrix} \theta-\theta_{bl} \\ \dot{\theta} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

The state of x(t) after backlash re-engagement can be computed by knowing that the gears fully re-engage:

$$x(t_{bl}) = \begin{bmatrix} \theta+\dot{\theta}\cdot t_{bl} \\ \dot{\theta}+\frac{T_m}{I_1}\cdot t_{bl} \\ \theta+\dot{\theta}\cdot t_{bl} \\ \dot{\theta} \end{bmatrix}$$

The time it took to re-engage the gears can be calculated by knowing the initial state and the end state:

$$\begin{bmatrix} \theta-\theta_{bl}+\dot{\theta}\cdot t_{bl} \\ \dot{\theta} \\ \theta+\dot{\theta}\cdot t_{bl} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} \frac{T_m}{2\cdot I_1}\cdot t_{bl}^2 \\ \frac{T_m}{I_1}\cdot t_{bl} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \theta+\dot{\theta}\cdot t_{bl} \\ \dot{\theta}+\frac{T_m}{I_1}\cdot t_{bl} \\ \theta+\dot{\theta}\cdot t_{bl} \\ \dot{\theta} \end{bmatrix}$$

The time it took to for the gears to re-engage is therefore:

$$t_{bl} = \sqrt{\frac{2\cdot\theta_{bl}\cdot I_1}{T_m}} \tag{3-1}$$

and the end state is:

$$x(t_{bl}) = \begin{bmatrix} \theta+\dot{\theta}\cdot\sqrt{\frac{2\cdot\theta_{bl}\cdot I_1}{T_m}} \\ \dot{\theta}+\frac{T_m}{I_1}\cdot\sqrt{\frac{2\cdot\theta_{bl}\cdot I_1}{T_m}} \\ \theta+\dot{\theta}\cdot\sqrt{\frac{2\cdot\theta_{bl}\cdot I_1}{T_m}} \\ \dot{\theta} \end{bmatrix}$$

The torque in the spring as a function of time can be calculated by using this state as an initial condition to the coupled model. The evolution of states is described by the following equation;

$$x(t) = e^{A\cdot t} \cdot x(t_{bl}) + \int_0^t e^{A\cdot\tau} \cdot B \; d\tau \tag{3-2}$$

To calculate the maximum torque the time at which it is maximal needs to be known. This time can be calculated by decomposing the equation into two parts. The natural response describes the evolution of states due to the initial condition. The driven response describes this evolution due to the torque that's applied to it. The maximum torque in the spring is a function of both parts which makes it difficult to compute the maximum torque. Fortunately both parts are sinusoids with the same frequency. This frequency depends on the second eigenvalue of the model. The first eigenvalue corresponds to the rigid-body motion and does not effect the toque in the spring. The second eigenvalue of the model is a function of the spring stiffness and both inertias;

$$w_1 = \sqrt{\frac{k \cdot (I_1 + I_2)}{I_1 \cdot I_2}}$$

In figure 3-2 the torque in the spring is visible. Also the two parts it is composed of are visible. From this figure two differences between the two parts are visible. One is that they

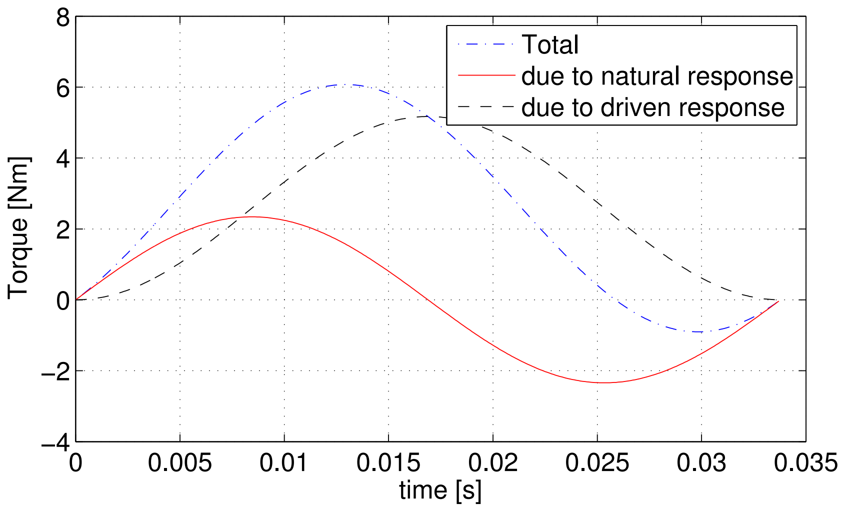


Figure 3-2: The torque in the spring as a function of time. The torque can be decomposed into two parts. One due to the initial condition and one due to the torque.

have different maxima and the other is that they have a different phase. The total torque can be parametrized by the following equation;

$$T_{total}(t) = T_{natural\ max} \cdot sin(w_1 \cdot t) + \frac{T_{driven\ max}}{2} \cdot (1 - cos(w_1 \cdot t));$$

If this equation is differentiated and set to zero. It can be solved for the time at which it is maximal. This time is:

$$t_{max} = \frac{2 \cdot \arctan \frac{T_{driven\ max} + \sqrt{4 \cdot T^2_{natural\ max} + T^2_{driven\ max}}}{2 \cdot T_{driven\ max}}}{w_1} \tag{3-3}$$

The maxima of both parts need to be calculated to find the time at which the combination of both parts is maximum.

Natural response

The natural response of the equation is:

$$x(t) = e^{A\cdot t} \cdot x(t_{bl})$$

The maxima of the natural part are at each $w_1(\frac{1}{4} + n)$ for $n \in \mathbb{N}$. The first maximum is at $w_1\frac{1}{4}$. This moment is used to calculate the maximum value which is at:

$$t_{max} = \frac{\pi}{2 \cdot w_1}$$

With this t_{max} the maximum torque of the sinusoid is:

$$T_{natural\ max} = \frac{k \cdot \sqrt{2 \cdot T_m \cdot I_1 \cdot \theta_{bl}}}{I_1 \cdot \sqrt{\frac{k}{I_1} + \frac{k}{I_2}}} \tag{3-4}$$

This equation can be compared to a single mass-spring system by taking the limit of I_2 to infinity. In this case the maximum becomes:

$$\lim_{I_2 \to \infty} T_{natural\ max} = \frac{k \cdot \sqrt{2 \cdot I_1 \cdot \theta_{bl}}}{I_1 \cdot \sqrt{\frac{k}{I_1}}} = \sqrt{2 \cdot k \cdot T_m \cdot \theta_{bl}} \tag{3-5}$$

The maximum torque for a single mass-spring system is straight forward to compute. The force in the spring will be at its maximum if all kinetic energy is converted into potential energy:

$$\frac{1}{2} \cdot I_1 \cdot \dot{\theta_1}^2 = \frac{1}{2} \cdot k \cdot \theta_1^2$$

The torque in this spring is proportional to θ_1 with a factor k. The speed of the first inertia is known so the torque can be solved:

$$\theta_1 = \sqrt{\frac{I_1}{k} \cdot \dot{\theta_1}^2}$$

$$\dot{\theta_1} = \frac{T_m}{I_1} \cdot \sqrt{\frac{2 \cdot I_1 \cdot \theta_{bl}}{T_m}}$$

$$T_{max} = k \cdot \theta_1 = k \cdot \sqrt{\frac{I_1}{k} \cdot \frac{2 \cdot T_m \cdot \theta_{bl}}{I_1}} = \sqrt{2 \cdot k \cdot T_m \cdot \theta_{bl}}$$

This results in the same equation as equation 3-5 and suggests that 3-4 accurately defines the maximum of the natural part.

Driven response

The equation of the driven response is:

$$x(t) = \int_0^t e^{A\cdot\tau} \cdot B \; d\tau$$

The maxima of this response are at each $w_1(\frac{1}{2} + n)$ for $n \in \mathbb{N}$. The first maximum is at $\frac{w_1}{2}$. This time is used to calculate the maximum:

$$t_{max} = \frac{\pi}{w_1}$$

If this time is used the following maximum torque is calculated:

$$T_{driven\ max} = \frac{2 \cdot I_2 \cdot T_m}{I1 + I2} \tag{3-6}$$

Also this result can be checked with a single mass-spring system:

$$\lim_{I_2\to\infty} T_{driven\ max} = \lim_{I_2\to\infty} \frac{2 \cdot I_2 \cdot T_m}{I1 + I2} = 2 \cdot T_m \tag{3-7}$$

Which is also the maximum torque when a single mass-spring system oscillates due to a torque.

The torque in the spring, $T_{max}(I_1, I_2, T_m, \theta_{bl}, k)$, now becomes:

$$T_{max} = \frac{I_2 - I_2 \cdot cos(w_1 \cdot t_{max}) + \sqrt{2 \cdot I_2 \cdot T_m \cdot \theta_{bl} \cdot k \cdot (I_1 + I_2)} \cdot sin(w_1 \cdot t_{max})}{I_1 + I_2} \tag{3-8}$$

Where $w_1 \cdot t_{max}$ is:

$$w_1 \cdot t_{max} = 2 \cdot \arctan \sqrt{\frac{2 \cdot (I_2 \cdot T_m + 2 \cdot I_1 \cdot \theta_{bl} \cdot k) I_2 \cdot T_m}{4 \cdot \theta_{bl} \cdot k \cdot (I_1 + I_2)}}$$

This equation can be simplified assuming that $I_1 << I_2$. In this case the limit of I_2 to infinity can be calculated:

$$\lim_{I_2\to\infty} T_{max} = T_m + \sqrt{T_m^2 + 2 \cdot T_m \cdot \theta_{bl} \cdot k} \tag{3-9}$$

For large values of $\theta_{bl} \cdot k$ this equation approximates equation 3-5. For small values of $\theta_{bl} \cdot k$ this equation approximates equation 3-7. This suggests that equation 3-8 and 3-8 accurately describe the total maximum torque occurring.

3-1-3 Maximum stress

According to [8] there is a linear relationship between the force on the teeth and the torque transmitted. This relationship is according to equation 21.67:

$$F_{max} = \frac{T_{max}}{d_w/2}$$

Here d_w is the effective diameter of the gear. This book also defines a linear relation between this force and the tooth base stress. According to equation 21.82 it is:

$$\sigma_{max} = \frac{F_{max}}{b \cdot M_n} \cdot Y_{Fa} \cdot Y_{Sa} \cdot Y_\epsilon \cdot Y_\beta \tag{3-10}$$

Here the Y factors are used to correct for approximations. In table 3-1 the values used for this equation are visible. Assuming that this tooth root stress is the maximum stress in the tooth then this stress can be related to mechanical failure.

3-1-4 Predicted MTBF

In [4] stress and failure are linked to each other. They are linked with each other in S-N curves. These curves represent the number of times a material can withstand a certain completely reversed cycle before it fails. In figure 3-4 these curves are displayed. In this figure also an approximation to these curves is visible. This approximation is performed by fitting a power relation to the curves.

$$\sigma_{ar} = \sigma'_f \cdot (2 \cdot N_f)^d$$

Here σ'_f and d are the fitting parameters. The teeth are made out of anodized aluminium. Assuming that this aluminium is 7075-T6 aluminium, the fitting parameters are 4402 [MPa] for σ'_f and -0.262 for d. These parameters cause the stress in the material as a function of maximum cycles to be overestimated for small number of cycles. The Dynamixel RX-28 gearbox has a MTBF of 5 minutes during the initial learning phase. This indicates that it is more important to have good fitting parameters at larger cycles numbers. The equation can be inverted to get the maximum amount of cycles as a function of a completely revered cycle.

$$N_f = \frac{1}{2} \cdot \left(\frac{\sigma_{ar}}{\sigma'_f}\right)^{\frac{1}{d}} \tag{3-11}$$

In order to predict failure the stress cycle in the gears due to backlash re-engagement must be linked to an equivalent completely revered cycle. This is a cycle which can be withstood an equal number of times and has its average at zero. In figure 3-3 the parametrization of the stress signal is displayed. In the gear teeth there are only positive stresses, therefore we can assume that the cycle starts at zero stress. The equivalent completely reversed cycle can be calculated by applying the Smith, Watson and Topper relationship [4].

$$\sigma_{ar} = \sqrt{\sigma_{max} \cdot \sigma_a} \tag{3-12}$$

b	M_n	d_w	Y_{Fa}	Y_{Sa}	Y_ϵ	Y_β
4 [mm]	0.4 [mm]	18.5 [mm]	2.15	2	1.22	1

Table 3-1: The values used to calculate the tooth base stress as a function of the torque on the gearbox.

Assuming that the stress cycle is a sinusoid, with a maximum of σ_{max} and a minimum of zero, its amplitude, σ_a, becomes half the maximum stress.

$$\sigma_a = \frac{\sigma_{max}}{2}$$

The equivalent completely reversed cycle therefore is:

$$\sigma_{ar} = \sqrt{\frac{\sigma_{max}}{2} \cdot \sigma_{max}} = \frac{1}{\sqrt{2}} \cdot \sigma_{max} \tag{3-13}$$

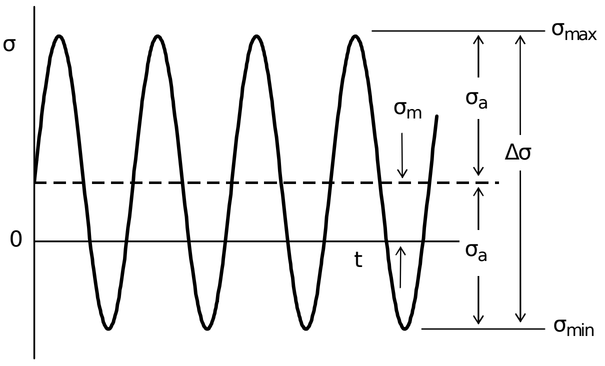


Figure 3-3: The signal parametrization from [4]

A cost function is created by inverting the number of cycles the material can withstand and summing them up:

$$J = \sum_{i=1}^{p} \frac{1}{45 \cdot N_{fi}} \tag{3-14}$$

Equation 3-9, 3-10, 3-11 and 3-13 are substituted in this equation and we assume that each of the 45 gear teeth is fatigued equally. In this equation p is the number of backlash re-engagement. The failure is predicted when J becomes larger than, or equal to, one. In the next section a strategy to increase the MTBF will be covered. This strategy will be tested on a Dynamixel RX-28 motor.

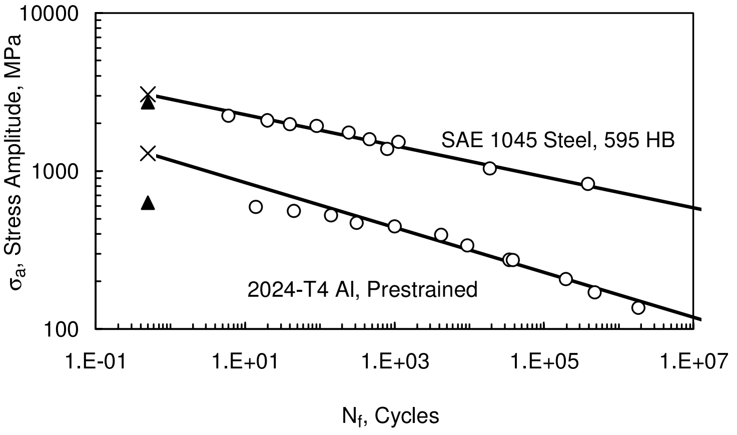


Figure 3-4: S-N curves from [4]. The number of times a cycle can be withstand before the material fails versus the stress amplitude of that cycle.

3-2 Mechanical failure of the dynamixel RX-28 gearbox

with physical testing two aspects of the MTBF prediction can be assessed. The accuracy of the MTBF prediction can be assessed and the effect of increasing the MTBF prediction on the actual MTBF can be assessed. First we will devise a strategy to increase the MTBF. Then we will design this experiment.

3-2-1 Increasing the predicted MTBF

In equation 3-15 we substitute equation 3-9, 3-10, 3-11 and 3-13 into equation 3-14.

$$J = \sum_{i=1}^{p} \frac{2}{45} \cdot \left(\frac{2 \cdot (T_{m_i} + \sqrt{T_{m_i}^2 + 2 \cdot T_{m_i} \cdot \theta_{bl} \cdot k}) \cdot Y_{Fa} \cdot Y_{Sa} \cdot Y_{\epsilon} \cdot Y_{\beta}}{\sqrt{2} \cdot dw \cdot b \cdot M_n \cdot \sigma'_f} \right)^{-\frac{1}{b}} \tag{3-15}$$

Most variables of this function are physical properties and cannot be changed by the learning algorithm. There are two variables that can be influenced by the algorithms. Those are the torque exerted by the motor while the gears re-engage and the number of backlash re-engagement. Reducing each of these parameters will result in an increased MTBF prediction. This prediction is much more sensitive to torques than to the number of backlash re-engagement.

One way of reducing both this torque and the number of backlash re-engagement is by limiting the change of torque. Limiting the change of torque is approximately equal to limiting the jerk in this case. Jerk is the derivative of acceleration with respect to time. According to [9] jerk limitation reduces actuator wear. An example of limiting the change of torque is displayed in figure 3-5. In this figure two signals with the same set of possible values are plotted. The set is {-1, -2/3, -1/3, 0, 1/3, 2/3, 1} times the maximum torque, which is equal for both signals. Both start by applying the maximum positive torque and try to have the lowest MTBF prediction. The random signal switches between maximum torque and minimum torque every time step. This causes the maximum amount of stress in the gearbox at every time step. The limited signal is not able to change as fast. Therefore, it takes a number of time steps before it reaches zero and even after it has started to switch between positive and negative torque it cannot switch with the same torque or frequency. The predicted MTBF

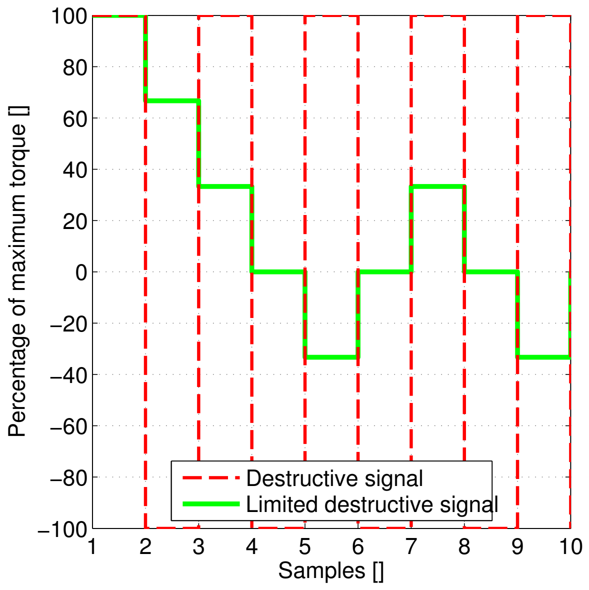


Figure 3-5: An example of how limiting the change of torque results in fewer backlash re-engagement and lower torques while re-engaging.

of the limited signal will be much larger compared to the MTBF of the random signal. The MTBF of the destructive signal is a lower bound for signals that have the same set of possible values and same limit on possible change. The effect of limiting the torque change on the learning process is covered more extensively in chapter 4. In this chapter the focus will be on looking at the relation between predicted and actual failure.

3-2-2 Predicted and actual MTBF

We will conduct an experiment to relate predicted and actual MTBF. This experiment can also be used to assess the contribution of random actions to the low MTBF when learning from scratch. In order to apply equation 3-15 we need to determine a number of parameters. These are the torque signal, θ_{bl} and k. Also that the gears re-engage each time the torque signal changes signal needs to be ensured. The experiment is designed to closely resemble the conditions of LEO hip gearbox. LEO's low MTBF is largely due to the gearbox of its hip motors failing. This way the contribution of of random motions to its low MTBF can be assessed. The limb inertia, I_2, and the reflected motor inertia, I_1, need to be calculated to look at the differences between predicting the MTBF based on equation 3-8 and 3-9. The experiment set-up is displayed in figure 3-6.

Stiffness

The stiffness between the limb and the reflected inertia, in the experiment, is defined by the stiffness of the aluminium link visualized in figure 3-6. The stiffness of this link is approximated to be 264 $[Nm \cdot rad^{-1}]$.

Backlash

According to [3] the backlash, θ_{bl}, is approximately 0.01 $[rad]$. This value will be used for each MTBF prediction.

Reflected motor and limb inertia

The test set-up is based on the inverted pendulum problem. It has one degree of freedom which has a mass connected to it via an arm. This one degree of freedom is powered by a Dynamixel RX-28.

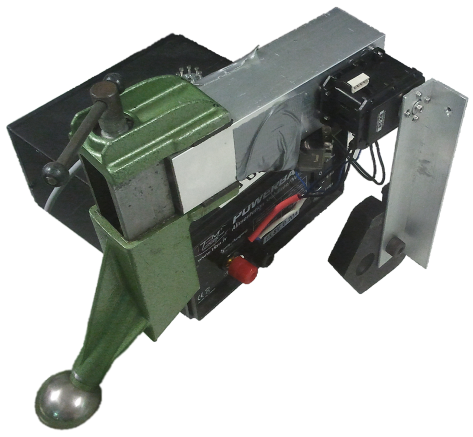


Figure 3-6: Test set-up to test the effect of toque change restrictions on the failure rate. The voltage source supplies 13.8 [V].

In [3] Leo is approximated as an eight-bar linkage. This linkage is displayed in figure 3-7. The dynamics of Leo's leg can be matched by a pendulum when Leo's leg is in its swing phase. During this swing phase the knee joint should be bended. This is needed to prevent foot scuffing. The knee angle, $-\varphi_{k,sw}$, is assumed to be bended $\pi/3$ $[rad]$. The angle between the foot and the lower leg, $\varphi_{a,sw}$, is assumed to be zero.

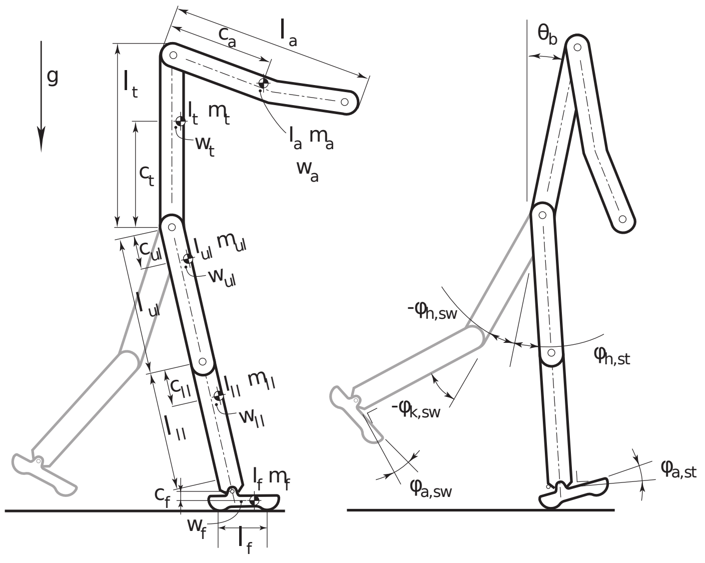


Figure 3-7: Schematic view of the model used by [3] to approximate Leo's dynamics.

In figure 3-8 a schematic of the pendulum is visible. The inertia of the pendulum can be calculated by looking at figure 3-7, applying the model conditions and using the parameters of [3].

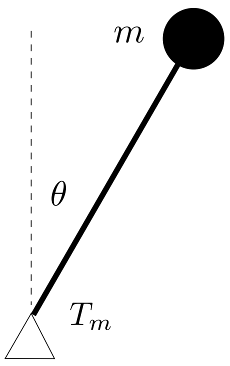


Figure 3-8: A visualisation of the inverted pendulum used.

$$I_{total} = I_{upper\ leg} + I_{lower\ leg} + I_{foot}$$

where;

$$I_{upper\ leg} = I_{ul} + C_{ul}^2 \cdot m_{ul}$$

$$I_{lower\ leg} = I_{ll} + (l_{ul} + \cos\varphi_{k,sw} \cdot C_{ll})^2 * m_{ll}$$

$$I_{foot} = I_f + (l_{ul} + \cos\varphi_{k,sw} \cdot (l_{ll} + C_f))^2 * m_f$$

The total inertia around the hip, I_{total}, is calculated to be 7.27 $[g \cdot m^2]$. Placing a mass of 500 $[g]$ on a distance of 0.12 $[m]$ from the joint creates an inertia of 7.20 $[g \cdot m^2]$, which is approximately equal to the total inertia around the hip. The pendulum has a stable equilibrium at the downward position. The equilibrium is stabilized by a torque that is generated by gravity. This creates a torque on the motor of at most 0.59 $[Nm]$. This is much smaller than the 3.2 $[Nm]$ the motor can generate therefore we do not model it.

In appendix A the properties of the Dynamixel RX-28 are given. The gearbox of this motor has a gear ratio of 1:193 and the motor inside the Dynamixel RX-28 is the Maxon 214897. This motor has an inertia of 0.868 $[g \cdot cm^2]$. The reflected inertia is calculated by multiplying the inertia of the motor by the gear ratio cubed.

$$I_1 = 193^2 \cdot 0.868 \cdot 10^{-3} \cdot 10^{-4} \; [kg \cdot m^2]$$

This results in a limb inertia which is a factor 2.2 larger than the reflected motor inertia.

Gear re-engagement

The gears have to re-engage fully every time the action signal changes sign. This is important for else one of the assumptions is violated. So even if the action is one third of the maximum voltage there should still be enough time to move θ_1 at least θ_{bl}. This puts restrictions on the sampling period. Assuming a linear relation between the input voltage and output torque the motor is able to supply a maximum of 3.2 $[Nm]$. The minimum torque at which backlash re-engagement occurs is therefore 3.2/3 $[Nm]$. These facts can be used to calculate the minimum sampling period needed with equation 3-1 assuming no friction in the gearbox.

$$t_{bl} = \sqrt{\frac{6 \cdot \theta_{bl} \cdot I_1}{T_m}} = \sqrt{\frac{6 \cdot 0.01 \cdot 0.868 \cdot 10^{-3} \cdot 10^{-4} \cdot 193^2}{3.2}} = 0.0078[s]$$

The gearbox does have friction but by putting the sampling period at 0.05 $[s]$, which is more than 6 times the minimal sampling period, gears will re-engage every time step.

Torque signals

The signal used in [3] is made out of seven actions. They are {-1, -2/3, -1/3, 0, 1/3, 2/3, 1} times the maximum voltage. in [3] a maximum voltage of 10.7 $[V]$ is used. We will use the same set but a different maximum voltage. The maximum voltage used in the experiment is 13.8 [V]. To test the effect of limiting the change of torque two different tests will be compared. In one test a random action signal will approximate the stress condition of the exploration phase. In the other test the change of torque will be limited which should increase the MTBF prediction.

The random signal would be able to produce the destructive signal of figure 3-5. The exploration is approximated by picking a torque out of the set pseudo-randomly every time-step. This way of creating the action signal does not take into account that the action signal becomes less random as the state-action values converge to their actual value. The MTBF can be calculated based on 1200 actions. A rough prediction of the MTBF can be calculated by applying equation 3-9 when calculating the maximum torque. In order to predict the MTBF a relation between the voltage and the torque T_m needs to be established. Both the random and the limited signal have a mean torque of approximately zero. Therefore we assume that during a backlash re-engagement the pendulum is close to its stable equilibrium and that the pendulum has a low angular velocity. So, we can state that there is no motion opposing voltage and V_{emf} is zero. This assumption together with equation 2-16 creates a linear relation between the voltage and T_m. Based on equation 2-16 the gain is 0.2488 $[Nm/V]$. Based on the Dynamixel documentation this gain is 0.2314 $[Nm/V]$ (see appendix A). Given that friction has not been considered during the modelling of the Dynamixel RX-28 we will use the lower gain.

This rough prediction of the MTBF is 99 $[s]$. I_1 and I_2 are known therefore equation 3-8 can be used to increase the accuracy of the prediction. The more accurate prediction of the MTBF is 298 $[s]$.

The limited signal starts at zero. From this point the signal is maintained, increased or decreased pseudo-randomly. The signal changes with $\pm 1/3$ of the maximum value. If the signal exceeds more than the maximum it is clipped back. The MTBF predictions of this signal can be calculated like the other signal. The rough MTBF prediction is 9769 $[s]$ and the more accurate prediction is 25934 $[s]$.

Hypothesis

There is a large difference between the MTBF predictions. The two-sample t-test is used to look at the probability that both means are equal. This probability is $1.07 \cdot 10^{-42}$ in case of the rough estimation and $1.11 \cdot 10^{-49}$ in case of the more accurate prediction. Limiting the change has a significant effect on the predicted MTBF of a signal. The hypothesis is therefore that by limiting the signal the actual MTBF of the gearbox will increase. To test this hypothesis the random signal is put on three gearboxes until they fail and the limited random signal is put on two gearboxes. It is difficult to automatically detect failure. Therefore, the test will have to be stopped regularly to manually inspect the gearbox for failure.

Results

There is a large difference between the MTBF of the gearbox during the random signal compared to the limited signal test. In table 3-2 the data from the random test is displayed. The mechanical states of the gearboxes after testing are displayed in figure 3-9. After the first two test several of the gear teeth are damaged such that the gearbox fails. After the third test only one teeth is damaged such that the gearbox fails. Some other teeth are also slightly damaged but are still able to function. The difference in damage could be caused by the fact that the last test is the shortest. The gearbox damage might have a self-reinforcing effect, causing more damage the gear as it increases. These tests result in a MTBF of 12180 $[s]$. This average has a confidence interval of 1245 $[s]$. In [3] the MTBF is much lower. This could be caused by the fact that falling and taking steps contribute more to the low MTBF or that the experiment does not represent the situation when learning from scratch.

The MTBF of the gearbox during the limited signal test could not be determined accurately. After 61620 $[s]$ the first test is stopped. Another test is stopped after 62280 $[s]$. In figure 3-10 the state of the gears after testing is visible. The MTBF of the gearbox during the limited signal test can be assumed to be larger than 61620 $[s]$.

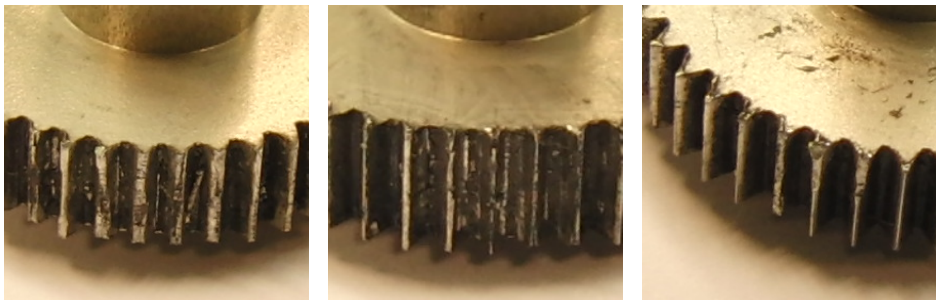

(a) Test 1 **(b)** Test 2 **(c)** Test 3

Figure 3-9: The state of the gears after the unlimited tests.

Test 1	11940 [s]
Test 2	13380 [s]
Test 3	11220 [s]
MTBF	12180 [s]
Confidence interval	1244 [s]

Table 3-2: The data from the random signal tests.

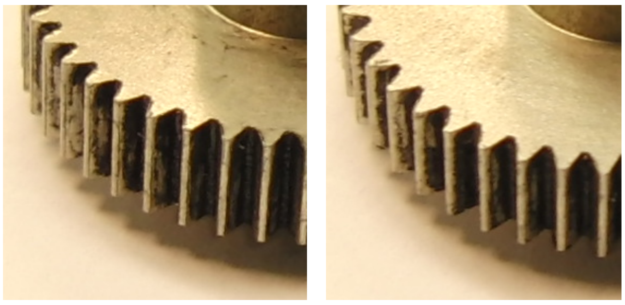

(a) Test 1 **(b)** Test 2

Figure 3-10: The state of the gears after restricted tests.

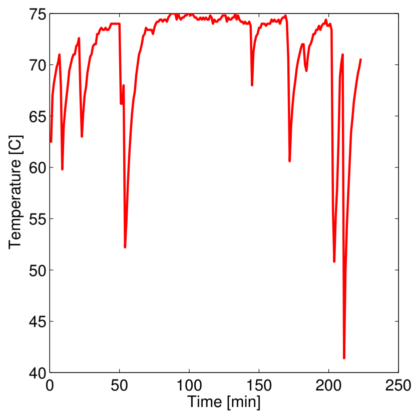
The two-sample t-test can be used to calculate the probability of the MTBF of the random test being 61620 [s]. This is $6.59 \cdot 10^{-4}$ therefore there is a significant difference between both tests. Limiting the change of torque increases the actual and predicted MTBF of the Dynamixel RX-28. Both the MTBF predictions underestimate the MTBF and the best prediction is still a factor of approximately 40 off.

The MTBF of 12180 [s] during the random test can be used to assess the contribution of the random motions to failure when learning from scratch. In LEO the stiffness k of the hip joint is 106 [$Nm \cdot rad^{-1}$] instead of the 264 [$Nm \cdot rad^{-1}$] in this experiment. In this experiment a maximum voltage of 13.8 [V] was used while the maximum voltage during learning is 10.7 [V]. This lower stiffness and voltage result in a higher MTBF prediction when learning from scratch. The actual MTBF is around 300 [s] which is lower instead of higher. Based on these facts we conclude that the contribution of the random motions to the low MTBF while learning from scratch is small. On the other hand failure from random motions could explain why the gearboxes of the hip motors fail most frequently. When taking the limb inertia into account the MTBF prediction increases. The limb inertias of the knee and ankle joints are smaller and could cause the larger MTBF. From this experiment it is clear that increasing the predicted MTBF increases the actual MTBF.

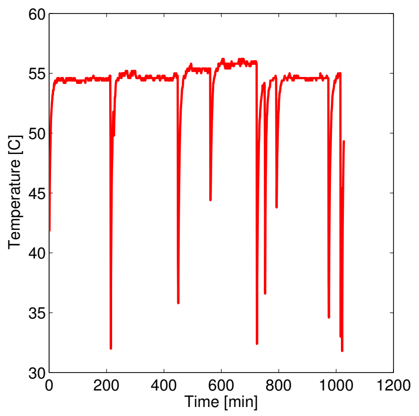
The MTBF prediction will be less complex by using equation 3-9 to calculate the maximum torque. This equation is used given that the increase of the predicted MTBF is interesting and not the predicted MTBF itself.

Motion opposing voltage

The assumption that there is no motion opposing voltage can be tested. The Dynamixel RX-28 measures its internal temperature and this measurement can be used to test this assumption. The system can only lose energy via heat transfer therefore, the internal temperature is a measure of power consumption. The temperature for one of the random tests is displayed in figure 3-11a. During the manual inspections the internal temperature of the



(a) The internal temperature of the Dynamixel RX-28 during one of the random signal tests.



(b) The internal temperature of the Dynamixel RX-28 during one of the limited signal tests.

Dynamixel decreases. The inspections can therefore be identified easily. The internal temperature of the Dynamixel stabilizes at approximately 75 $[°C]$. This temperature is close to the overheating temperature of 80 $[°C]$.

In figure 3-11b the temperature during one of the limited tests is displayed. The internal temperature stabilizes at approximately 55 $[°C]$. The internal temperature during a random signal is significantly higher. This indicates a higher power consumption. The average absolute torque, that the motor delivers each time step, is 1.79 $[Nm]$ in the case of the limited signal and 1.82 $[Nm]$ in the case of the random signal. These torques are calculated by the linear relation between torque and voltage. The power consumption should therefore be approximately equal which is not the case. This indicates that the model used to predict the MTBF might be too simplistic. Specifically the assumption that there is no motion opposing voltage, V_{emf}, seems to be invalid. This seems to be invalid based on the observation that during the limited test the motions consists of lower frequency but higher amplitude movements compared to the random test. Given that the MTBF failure is a rough prediction this assumption is still applied in the rest of the thesis. In the following chapter the learning performance of various algorithms that increase the predicted MTFB will be assessed. The rough prediction of the MTBF will be used.

Chapter 4

PADA

From a hardware point of view it is relatively easy to implement a limitation to the change of torque, thereby increasing the predicted MTBF. From a learning point of view however, it is more complicated. The learning performance can drop significantly if the predicted MTBF is increased. There are two main reasons for this: The limitation might prevent an optimal policy and the suboptimal policy will result in a lower end performance. The other is that the Markov property of the environment might be lost. In this chapter the learning performance of various algorithms that increase the predicted MTBF are assessed.

4-1 Algorithms

In this chapter the learning performance of six algorithms and their MTBF are assessed. The first of the algorithms is a SARSA(λ) algorithm. This algorithm will be used as a benchmark for the other algorithms which increase the MTBF by limiting the change of the action signal.

SARSA(λ)

The SARSA(λ) algorithm can be used for on-line, on-policy learning. The algorithm that is used has a learning rate (α) of 0.2, a discount rate (γ) of 0.99, an exploration rate (ϵ) of 0.05 and a trace decay rate (λ), of 0.92. These learning parameters will also be used by the other algorithms. This algorithm applies tile coding in order to speed up learning. The properties of this algorithm will be used as a benchmark for the other algorithms.

Low-pass filter

The change of torque can be limited by low-pass filtering the action signal provided by the SARSA(λ) algorithm before it is applied to the environment. The low-pass filter algorithm applies this. A low-pass filter is an intuitive method to prevent high frequency random motions. This low-pass filter could be implemented with a discrete first order filter. The equation of this filter is given by:

$$a_{filtered_k} = \alpha \cdot a_k + (1 - \alpha) \cdot a_{filtered_{k-1}} \tag{4-1}$$

A first order filter is used to keep the dynamics of the filter as simple as possible. This filter converts the discrete action signal into a continuous signal. When α goes to zero, the signal is unable to change and the predicted MTBF will be infinitely large. Learning the agent to balance the pendulum will be impossible if the signal is unable to change. An α of one corresponds to no filtering. In this case the algorithm is equal to the SARSA(λ) algorithm. The effect of different α values need to be investigated. The Markov property of the environment is lost when using a low-pass filter. This is caused by the fact that the previous applied torque is not known to the agent.

Markov low-pass filter

The Markov property of the environment can be preserved in the case of low-pass filtering by adding another state variable that represents the previous torque. The Markov low-pass filter applies this. A disadvantage of adding this state variable is that the number of state dimensions increase. This causes the learning speed to decrease. The advantage of preserving the Markov property could be undone by this disadvantage.

Integrating controller

Another method of limiting the change of torque can be implemented by looking at Delta modulation. Delta modulation is a method to digitize an analogue signal. In [10] a bit at each step defines whether the signal is increased or decreased with a fixed value. This digital signal of bits can be transformed back to an analogue signal by a delta demodulator. This delta demodulator is an integrator which gets fed by a bipolar signal, $\pm\ \Delta$. An example of a signal that could have been produced by a delta demodulator is the limited signal of figure 3-5. The integrating controller algorithm uses an integrating controller that closely resembles this delta demodulator [11]. An integrating controller is an integrator that integrates the action signal of the agent before it is applied to the environment. The integrator is fed a signal of $\pm\Delta$ or zero. A SARSA(λ) algorithm learns which of these three actions to apply in each state. The integrated signal has a fixed value, Δ, at which it can change value. With a delta of zero the signal cannot change and the MTBF increase to infinity but the agent will not be able to learn in this case. The Markov property of the environment is lost when using an integrating controller. This is caused by the fact that the previous applied torque is not known to the agent.

Markov integrating controller

The Markov property of the environment while integrating the actions from the agent can be preserved by adding the previous torque to the state. The Markov integrating controller applies this. Increasing the number of state dimensions, as with the Markov low-pass filter, might cause the learning speed to decrease.

PADA

So far each algorithm either loses the Markov property of the environment or increases the number of state dimensions to increase the MTBF. The Markov property can be preserved without adding a state dimension by performing the action integration differently.

An important observation for this fact is that while working with state-action values. Action selection can be a function of the previous action. The state-action value is defined as:

$$Q^\pi(s,a) = E_\pi\left\{R_t | s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+1+k} | s_t = s, a_t = a\right\}.$$

In other words the value of a state action is the future reward of being in a certain state, taking a certain action and following a certain policy after that. Due to the Markov property the future states and actions only depend on the current state and action. The evolution of the states is a Markov chain:

$$s_{t+1} = f(s_t, a_t)$$

$$a_{t+1} = g(s_{t+1}) = g(f(s_t, a_t))$$

Here the f represents the state transition function and g represents the policy. These functions can be used to iterate the states and actions to infinity. Thereby computing the future reward. The optimal state-action value is given by:

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right]$$

In this case the action is only a function of the next state but it does not have to be so. The next action can also be a function of the previous action:

$$s_{t+1} = f(s_t, a_t)$$

$$a_{t+1} = h(s_{t+1}, a_t) = h(f(s_t, a_t), a_t) = z(s_t, a_t)$$

This process is still a Markov chain which can be iterated to compute the future reward. The optimal state-action value now becomes:

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in f(a)} Q^*(s', a')\right]$$

The fact that the next action can be a function of the previous action can be used to perform action integration without losing the Markov property or increasing the number of state dimensions. The previous action dependent action algorithm, or PADA algorithm, uses this. The dependency of the next action to the current action is equivalent to the current action depending on the previous action:

$$a' \in f(a) \mapsto a_{t+1} \in f(a_t) \mapsto a_t \in f(a_{t-1})$$

Therefore the value of the current state becomes a function of the previous action:

$$V^*(s, a_{t-1}) = \max_{a \in f(a_{t-1})} Q^*(s,a)$$

The integrator can be implemented by applying SARSA(λ) with a different action selection. Normally the action with the highest state action value is selected:

$$a_t = g(s_t) = arg \max_a Q(s_t, a)$$

It can be implemented by only evaluating the actions that are on a fixed distance from the previous action and evaluating the previous action itself:

$$a_t = g(s_t, a_{t-1}) = arg \max_{a \in f(a_{t-1})} Q(s_t, a) \tag{4-2}$$

Where $f(a_{t-1})$ is:

$$f(a_{t-1}) = \{a |\ a - a_{t-1} \in \{\pm\Delta, 0\}\} \tag{4-3}$$

From these three state-action values the highest is selected. The future only depends on these three actions, therefore the Markov property is preserved. This is comparable with an integration controller because each time the agent still decides whether to maintain, decrease or increase. It is different from an integration controller because it makes this decision based on the expected reward of the actual action taken instead of the expected reward of maintaining, decreasing or increasing. Switching between the PADA algorithm and the SARSA(λ) can be done by starting to evaluate every action instead of three ($f(a) = A$).

4-1-1 Algorithm properties

Increasing the MTBF will cause the learning performance to change. The actual MTBF is roughly estimated by Equation 3-14 therefore it is difficult to optimise between increasing it and decreasing learning performance. The objective is therefore defined as the 95% confidence that the average performance is less than 33% decreased, while increasing the MTBF.

End performance

The performance of the algorithm is analysed by performing a test run with an interval of 10 runs. During this test run the exploration rate is set to zero therefore all actions are based on the greedy policy. The rewards given in this test run can be summed up to give a measure of performance. The end performance is computed by looking at the average performance of the last 10 % of the test runs.

Learning speed

The increase of performance during learning can be approximated by an exponent which asymptotically goes to the end performance. The rate at which it converges is defined by $1 - e^{-1/\tau}$ where τ is the time constant. The time constant of a trial is computed by looking at when $1 - 1/e^2 = 0.865 = 86.5\%$ of the end performance is reached and dividing this by two. Averaging the time constants of the trials gives the average time constant. Unfortunately the performance can be a noisy signal. To limit the influence of this noise the performance signal is filtered. The signal is filtered with:

$$y_k = 0.3375 \cdot (x_k + x_{k-1}) + 0.3249 \cdot y_{k-1}$$

From this smoothed signal the learning speed is obtained.

4-2 Simulated inverted pendulum environment

The MTBF and learning performance can be investigated by simulating how each of the algorithms learn to swing up an inverted pendulum to its unstable equilibrium.

4-2-1 Inverted pendulum simulation

The inverted pendulum environment is one of the most used environments to benchmark reinforcement learning algorithms. Here the environment is used to investigate the learning capability and MTBF of various algorithms. A visualisation of the model used is given in figure 3-8. The pendulum consists of a point mass, of 2 [kg], which is located 0.2 [m] from the joint and its moment of inertia is 0,048 $[kg \cdot m^2]$. The torque on the pendulum is between ± 1.5 $[Nm]$ and assumed to be generated by the same hardware as in LEO. The state of this system is sampled at 20 [Hz] and it consists of the angle θ and the speed $\dot{\theta}$ of the pendulum. Its reward function is $-5\theta^2 - 0.1\dot{\theta}^2 - a^2$, with a the applied torque. The state-action values are optimistically initiated by setting their values to zero. Each of the algorithms will learn three actions. Their averages are computed over 29 trials.

4-2-2 Results

SARSA(λ)

The learning performance of the SARSA(λ) algorithm is displayed in figure 4-1. The agent cannot increase its performance after 20.000 $[steps]$. The average performance of the algorithm converges to -851. This average has a 95% confidence interval of 39. 33% of -851 is approximately -281 therefore the average end performance of other algorithms should be above -851-281+39=-1093 with a confidence of 95%. The average time constant of it is 7860 $[steps]$ or 393 $[s]$. This average has an confidence interval of 520 $[steps]$ or 26 $[s]$, therefore the average time constants of the other algorithms should be below 498 $[s]$. The algorithm has an MTBF of 1596 $[s]$ with a confidence interval of 34 $[s]$.

Other algorithms

In fig. 4-2 the MTBF of the other algorithms are displayed as a function of either the α or Δ respectively. All algorithms are able to significantly increase the MTBF and have small confidence intervals. In figure 4-3 the end performance of the algorithms as a function of the MTBF is visualized. The plain integrating controller algorithm is the only one that is unable to achieve equivalent end performance. As shown in figure 4-4 the time constant generally increases when the MTBF is increased. Both algorithms that increased the number of state dimensions are unable to achieve equivalent time constants. This leaves two algorithms which are able to reach the objective. The low-pass filter algorithm with an α of 0.834 or larger and the PADA algorithm with a Δ of 0.5. Both these algorithms are able to increase the MTBF by a factor of two. The PADA algorithm with this delta increases the MTBF by reducing the number of backlash re-engagements during exploration while the low-pass filter decreases the torque at which it re-engages.

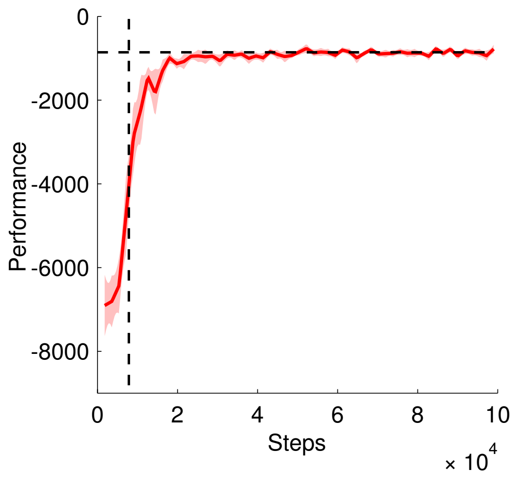


Figure 4-1: The learning performance of a SARSA(λ) algorithm. The mean and the confidence interval is plotted. Also average end performance and time constant are displayed.

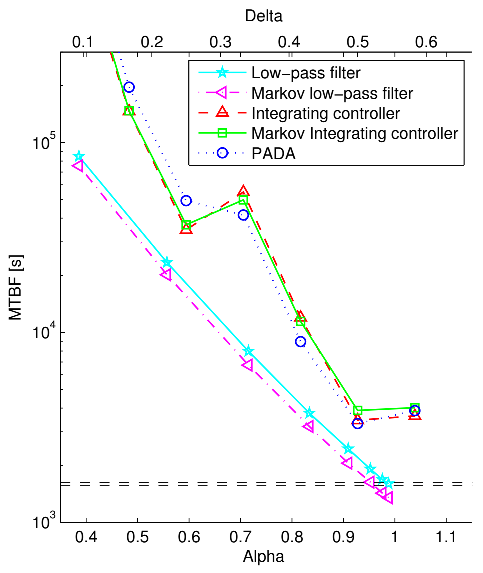


Figure 4-2: The MTBF can be increased by decreasing the Δ or α parameter. The horizontal dashed lines represent the confidence interval of the average MTBF of the SARSA(λ) algorithm.

4-3 Physical environment

In simulation the PADA algorithm and low-pass filter algorithm can have equivalent learning performance as the SARSA(λ) algorithm. The MTBF can be doubled by applying the PADA algorithm. The simulated pendulum does not suffer from noise. In this section the performance of the PADA and low-pass filter algorithm is compared to that of the SARSA(λ) while learning on a physical environment. To test the performance of these algorithms on a physical environment an inverted pendulum is used. The physical inverted pendulum can be seen in figure 4-5. This inverted pendulum is used in [12] to investigate how model learning can speed up the learning process. This pendulum has the same reward function and optimistic

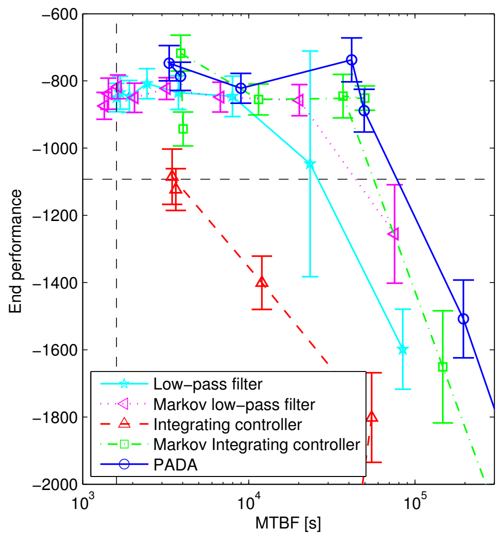


Figure 4-3: Increasing the MTBF generally has a decreasing effect on the end performance. The integrating controller algorithm is unable to achieve an equivalent end performance. The horizontal dashed line is the minimal end performance that should be reached with a 95 % certainty. The vertical dashed line is the MTBF of the SARSA(λ) algorithm.

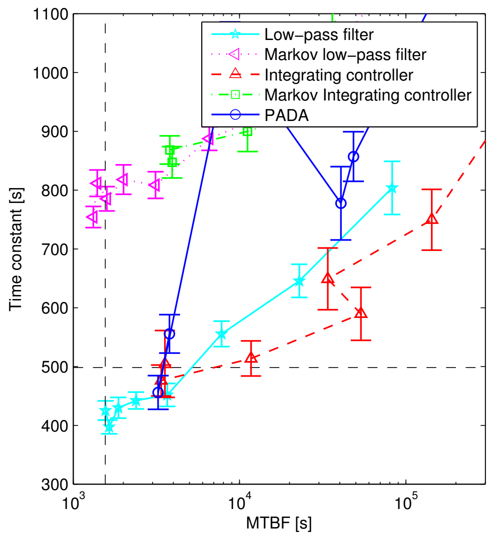


Figure 4-4: Increasing the MTBF generally has a increasing effect on the time constant. Both the algorithms that increased the state to preserve the Markov property are unable to achieve equivalent learning speed. The horizontal dashed line is the maximal time constant that should be reached with a 95 % certainty. The vertical dashed line is the MTBF of the SARSA(λ) algorithm.

initialisation as the simulated inverted pendulum only the sampling period is 0.03 instead of 0.05 [s] and the action supplied to this environment is the voltage on the motor instead of a torque. The physical properties of this environment are given in [12]. As before we will assume no motion opposing voltage. In this case there is a linear relation between torque and voltage of $5.64 \cdot 10^{-3}$ $[Nm/V]$

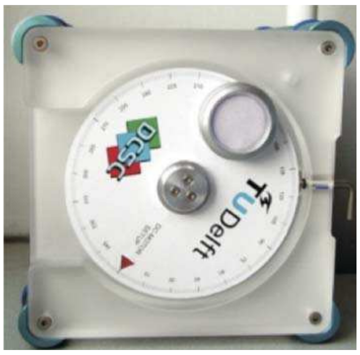


Figure 4-5: The physical inverted pendulum.

4-3-1 Tests

The performance of a standard SARSA(λ) algorithm will be compared to a PADA algorithm with a Δ of 0.5 and a low-pass filter with an α of 0.834. Each algorithm will have an action set of three actions; -3, 0, 3 $[V]$. Assuming a linear relationship between voltage and torque, and that it is generated by the same hardware as in LEO, the MTBF can be calculated. The averages are computed over 28 trials.

4-3-2 Results

MTBF

The MTBF is based on the first 1200 steps or 36.4 $[s]$ of the trials. The SARSA(λ) algorithm has a MTBF of $6.92 \cdot 10^7$ $[s]$, the low-pass filter algorithm a MTBF of $1.31 \cdot 10^8$ $[s]$ and the PADA algorithm a MTBF of $1.39 \cdot 10^8$ $[s]$. The PADA algorithm and the low-pass filter algorithm are both able to increase the MTBF by a factor of two. As stated in the previous section the PADA algorithm reduces the number of backlash re-engagement while the low-pass filter decreases the torque at which it re-engages. This is visible in figure 4-7 where the lower torque results in a smaller increase of J at each backlash re-engagement. From this figure it is also clear that the accumulation of J reduces after a few minutes. This indicates that the policy is relatively close to being optimal.

learning performance

In figure 4-6 the average performance as a function of the number of steps is visualized. All algorithms have equivalent end performance compared to SARSA(λ). From this figure it is clear that the PADA algorithm learns significantly faster than the other two. This indicates that there are cases in which the PADA algorithm does not have to sacrifice performance in order to increase the MTBF.

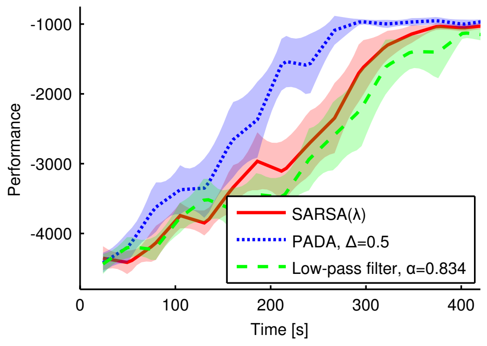


Figure 4-6: Average learning curve and confidence interval of the physical inverted pendulum.

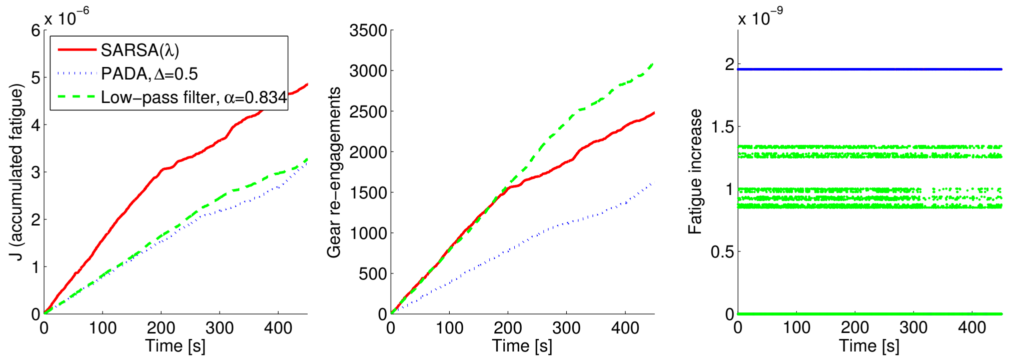


Figure 4-7: The accumulated fatigue, the number of backlash re-engagement and the values at which the fatigue increases plotted for one trial. It is based on the first 40 $[s]$ of the trial. The MTBF is predicted by predicting at what time J reaches the value of one. The PADA algorithm increases this prediction by reducing the number of backlash re-engagement while the low-pass algorithm reduces the increase of J at each backlash re-engagement

Chapter 5

LEO

The learning process on LEO can be simulated to assess the properties of both MTBF increasing algorithms on complex tasks relative to SARSA(λ). The model used to simulate LEO is visualized in fig. 5-1.

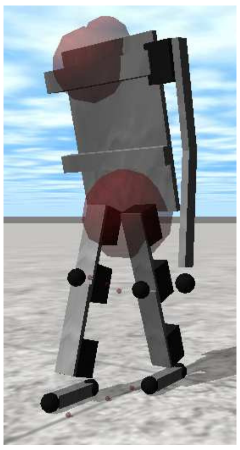


Figure 5-1: A model of LEO used to simulate the learning process on the physical system.

It has 10 state dimensions and is highly non-linear. The algorithm properties are computed by averaging over 15 trials. The SARSA(λ) algorithm learns to pick the optimal voltage, out of the set $\{-1,\ -2/3,\ -1/3,\ 0,\ 1/3,\ 2/3,\ 1\} \cdot 10.7\ [V]$, to get the optimal walking gait. It learns this for both the two hips and the swing leg knee joint therefore the number of action dimensions is three. The displacement of the swing foot is used for the reward function. The agent is rewarded 300 $[m^{-1}]$ for positive displacement and -300 $[m^{-1}]$ for negative displacement. To promote walking speed the agent is rewarded -1 $[s^{-1}]$ each time step and to increase energy efficiency the energy usage is rewarded -3 $[J^{-1}]$. The agent is rewarded -125 each time it falls. The total reward function is a summation of these rewards. The state-action values are optimistically initiated by setting their values between 0 and 0.01.

Small Δ or α values will cause the learning performance to drop, making them undesirable. The predicted MTBF of an algorithm is assumed to be large in the exploration phase relative to the optimization phase. In figure 5-2 the accumulation of fatigue is visualized for one trial. In this trial, after 5000 [s] of simulated learning, this accumulation decreases. This indicates that the exploration phase is over and that the policy is close to being optimal. In

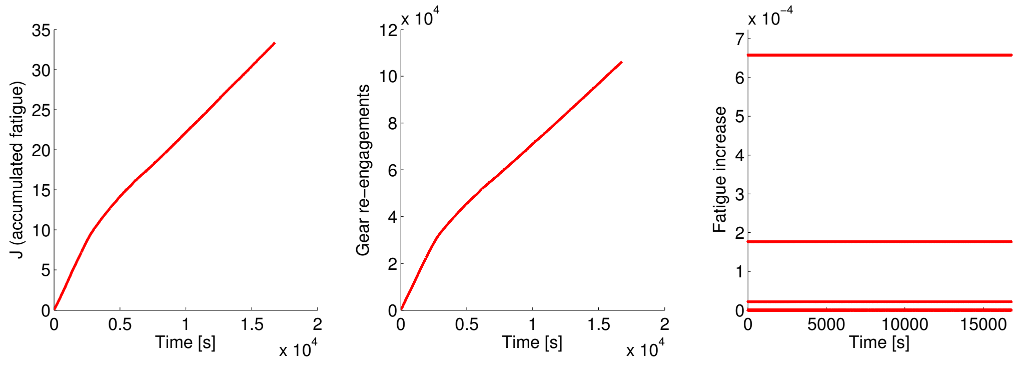


Figure 5-2: The accumulated fatigue during a learning trial. The algorithm used was SARSA(λ). After the exploration phase the accumulation of fatigue decreases.

order to apply small values of Δ or α without a significant drop in performance they could be used during the exploration phase only. After the exploration phase (7333 [s] in LEO's case) the algorithms can be switched back to SARSA(λ). In most trials the robot has already found a sub-optimal walking gait at this point. This is important for the PADA algorithm because when the value of each action is evaluated instead of three, one of these three should still have the highest value. Else, random motions due to optimistic initialisation occur after switching which results in a drop of performance. The SARSA(λ) algorithm optimizes this gait for another 7333 [s]. Cutting up the learning process into parts changes the apparent system dynamics (which may require some relearning) but it has advantages when looking at the MTBF. In nature many learning processes are also made out of parts. For instance rat foetuses already start to learn motions while they are still in their mothers womb [13].

SARSA(λ)

The SARSA(λ) algorithm has an average end performance of 1760 and a confidence interval of 262. The average end performance of the other algorithms should be be above 1441. The algorithm has a MTBF of 362 [s].

Other algorithms

As can be seen in fig. 5-3, the MTBF can be increased further when both algorithms are switched back after 7,333 [s]. In fig. 5-4 the learning curves of a PADA and low-pass filter which both have the same MTBF are displayed. These two algorithms both increase the MTBF by a factor of 108 to approximately 39,000 [s]. Also the SARSA(λ) algorithm is displayed as a reference. While the PADA algorithm is able to learn with small values of Δ and switch back to SARSA(λ) without losing performance the low-pass filter is not. Smaller values of Δ might also be possible when applying the PADA algorithm. The PADA algorithm is clearly better at keeping the learning performance equivalent while increasing the MTBF.

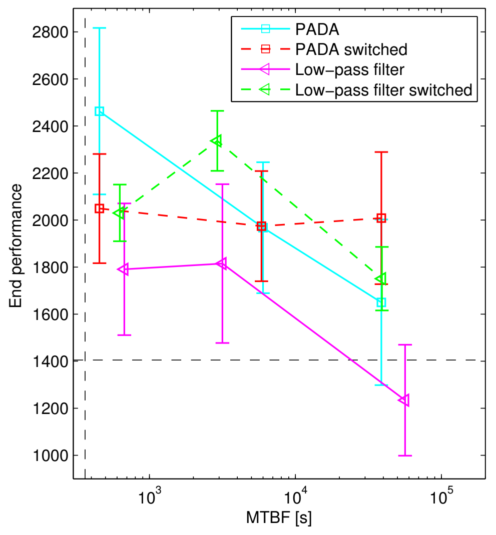


Figure 5-3: Increasing the MTBF generally has a decreasing effect on the end performance. This effect can be mitigated by switching back to SARSA(λ) after the exploration phase.

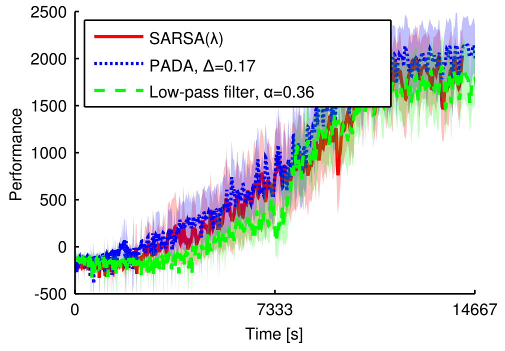


Figure 5-4: Average learning curve of the simulated learning process of LEO. The SARSA(λ) learning curve is displayed as a reference. The other two methods have a MTBF of 39,000 $[s]$.

Falling during learning

In figure 5-5 and figure 5-6 the average fall rates of LEO for the SARSA(λ) and the two MTBF increasing algorithms while learning and during test runs are visualised. In this figure the confidence intervals overlap which suggests that there is no significant difference between each algorithm. The differences that occur are still interesting. The low-pass filter seems to have a lower fall rate than the SARSA(λ) algorithm, The PADA algorithm also has a lower fall rate after about 2450 $[s]$ of learning. After both algorithms are converted to SARSA(λ) it is the low-pass filter which has the lower fall rate. This contradicts with the observation of the performance. This contradiction can be explained when looking at the fall rates during the test runs. during the test runs it is the PADA algorithm that has the lowest fall rate. This indicates that it is the exploration that causes the increase in the fall rate after both methods are converted back to SARSA(λ). The PADA algorithm has not learned to recover due to certain random motions while the low-pass filter has.

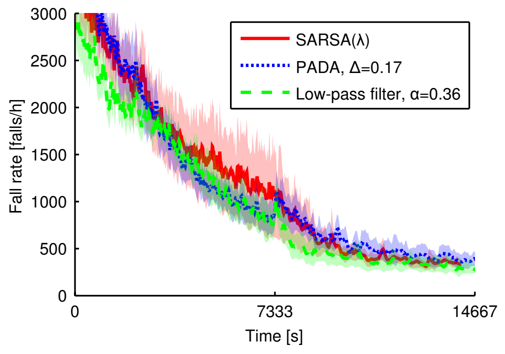


Figure 5-5: Average fall rates of the simulated learning process of LEO. The SARSA(λ) rate is displayed as a reference. The other two methods have a MTBF of 39,000 $[s]$.

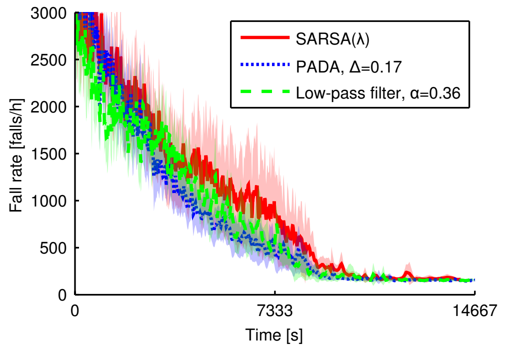


Figure 5-6: Average fall rates of the simulated learning process of LEO while only taking greedy actions. The SARSA(λ) rate is displayed as a reference. The other two methods have a MTBF of 39,000 $[s]$.

Chapter 6

Discussion

The prediction of a MTBF relies heavily on a number of assumptions. Some of these assumptions have a known effect on the prediction. For instance, Assuming that the reflected motor inertia is much larger than the limb inertia result in a lower prediction. Other assumptions have an unknown effect. Assuming a linear relation between voltage and torque has an unknown effect on the prediction. These assumptions lead to a rough prediction of the MTBF. The accuracy of the MTBF can be increased by taking more effects into account but the complexity of the problem inhibits a very accurate prediction. Therefore, there is only one way to test if the MTBF of LEO is increased during learning and that is by applying the PADA or low-pass filter algorithm to LEO in a physical test. The result of this test would be interesting but unfortunately it could not be conducted because of the delicate hardware in LEO.

If the accuracy of the MTBF prediction due to random motions is increased, and the effects of falling and taking steps are also taken into account, the learning process could be adapted to maximize the MTBF of the entire process. This would be more interesting than the maximisation of the MTBF in the exploration phase while keeping the learning process approximately equivalent but increasing the MTBF by decreasing random motions will be a part of this optimisation.

In order to maximize the MTBF in the exploration phase it is efficient to increase it in this phase specifically. Especially the PADA algorithm is able to do this efficiently. A problem with applying this is that the transition from the exploration phase into the optimisation phase is not defined in this thesis. Instead, in chapter 5 we assume that this phase is over after 7333 $[s]$. This assumption does not take into account that each trial has a different exploration phase duration. It would be interesting to see the learning performance of an agent that starts by learning with an algorithm that has a high MTBF and switches to an algorithm that has a lower MTBF when it detects that the exploration phase is over.

The PADA algorithm has a higher potential to increase the predicted MTBF in case of LEO. This is mostly due to the fact that there are seven, equally distributed, possible actions for the robot to take. This causes the Δ of the PADA algorithm to be 1/6 as minimal value, given that the algorithm needs to be able to switch back to SARSA(λ) and excluding a Δ of zero. The α value of the low-pass filter algorithm can be set independently of the actions therefore this algorithm can increase the MTBF to any value higher than SARSA(λ). The PADA algorithm can be adapted such that after the exploration phase the action set changes. This would enable the Δ value to be set independently of the action set but the effect of this is not covered in this thesis.

In this thesis the learning properties and MTBF of SARSA like algorithms are studied. Many other types of reinforcement learning methods exist which are applied to physical systems. The learning performance of these other types while increasing the MTBF are also interesting but they were not covered in this thesis.

One of these other methods severely limit the policy to increase learning speed and prevent random motions. One could argue that the PADA algorithm does basically the same thing without increasing the learning speed but that several properties of the PADA algorithm would be neglected. For instance, Its ability to switch back to a SARSA(λ) algorithm or the amount of policies it could learn.

It would also have been interesting to look at the fall rate of the PADA learning process if after it has been converted back to SARSA(λ) the exploration would still be done based on the PADA algorithm. This would probably preserve the low fall rate but it could effect the learning speed negatively.

Chapter 7

Conclusion

The MTBF is roughly predicted due to a number of assumptions therefore it is the increase of this prediction that is interesting instead of the value itself. Increasing this predicted MTBF also increases the actual MTBF. The learning algorithm should be adapted such that the predicted MTBF increases, while maintaining equivalent learning performance compared to SARSA(λ).

The PADA and low-pass algorithm are both able to increase the MTBF by decreasing random motions, without negatively influencing the learning performance. The PADA algorithm does not violate the Markov property during learning. The low-pass filter algorithm does, which is a disadvantage. The PADA algorithm has more potential to increase the MTBF than the low-pass filter when applied to more complex systems. This is due to the PADA algorithm, which can be switched to the SARSA(λ) algorithm without losing performance. The PADA algorithm can increase the MTBF by a factor of 108 in case of LEO. In some cases the PADA algorithm can even out-perform the SARSA(λ) algorithm.

From an experiment it is shown that the random motions contribute to the low MTBF and that increasing the predicted MTBF also increases the actual MTBF. From simulating learning processes and physical learning processes it is clear that this predicted MTBF can be increased without negatively effecting the learning process. From these facts it can be concluded that it is possible to reduce the MTBF of robotic systems by decreasing random motions due to learning, without negatively effecting the learning process.

Appendix A

Dynamixel datasheet

1-3. Specifications of RX-28

	RX-28	
Weight (g)	72	
Dimension (mm)	35.6 x 50.6 x 35.5	
Gear Reduction Ratio	1/193	
Applied Voltage (V)	at 12V	at 16V
Final Reduction Stopping Torque (kgf.cm)	28.3	37.7
Speed (Sec/60 degrees)	0.167	0.126

Resolution	0.29°
Running Degree	300°, Endless Turn
Voltage	12V~16V (Recommended voltage: 14.4V)
Max Current	1200mA
Running Temperature	-5℃ ~ +85℃
Command Signal	Digital Packet
Protocol	RS485 Asynchronous Serial Communication (8bit,1stop, No Parity)
Link (Physical)	RS485 Multi Drop Bus
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Sensing & Measuring	Position, Temperature, Load, Input Voltage, etc.
Material Quality	Full Metal Gear, Engineering Plastic Body
Motor	Maxon RE-MAX
Standby Current	50 mA

Appendix B

IROS Paper

Learning while preventing mechanical failure due to random motions

H. J. Meijdam, M. C. Plooij and W. Caarls

Abstract— **Learning can be used to optimize robot motions to new situations. Learning motions can cause high frequency random motions in the exploration phase and can cause failure before the motion is learned. The mean time between failures (MTBF) of a robot can be predicted while it is performing these motions. The predicted MTBF in the exploration phase can be increased by filtering actions or possible actions of the algorithm. We investigated 5 algorithms that apply this filtering in various ways and compared them to SARSA(λ) learning. In general, increasing the MTBF decreases the learning performance. Three of the investigated algorithms are unable to increase the MTBF while keeping their learning performance approximately equal to SARSA(λ). Two algorithms are able to do this: the PADA algorithm and the low-pass filter algorithm. In case of LEO, a bipedal walking robot that tries to optimize a walking motion, the MTBF can be increased by a factor of 108 compared to SARSA(λ) learning. This indicates that in some cases, failures due to high frequency random motions can be prevented without decreasing the performance.**

I. INTRODUCTION

With robotic systems assisting humans in every day life, it is increasingly important for systems to be able to autonomously adapt to new situations. The field of Reinforcement learning contributes to this autonomous behaviour. Reinforcement learning can be applied to let physical systems learn to perform a motion without having prior knowledge of this system. During the learning process large stresses occur in these physical systems. They need to be able to withstand these stresses order to prevent failure.

The bipedal robot LEO (see fig. 1a) was designed to withstand large stresses while learning process an optimal gait autonomously with little or no prior knowledge. Unfortunately the robot fails too quickly for it to learn a stable gait from scratch. During the learning process there are three main causes of failure: taking steps, falling and random motions. The first two causes are essential parts of the learning process which can only be reduced by faster learning or redesigning the robot. The third is specific to the learning algorithm used. Failure occurs in the system due to high frequency random motions in the early learning phase. These motions are not an essential part of the learning process. Therefore it should be able to reduce failure without significantly influencing the learning process.

The rest of this paper is structured as follows. In section II other reinforcement learning experiments with physical robots are covered, specifically how they prevent damage from exploration. Reinforcement learning is covered in section III. In section IV the MTBF of LEO's gearboxes are linked to action signals. Various algorithms that could be used to increase the MTBF are presented in section V. The performance of these algorithms is quantified in section VI.

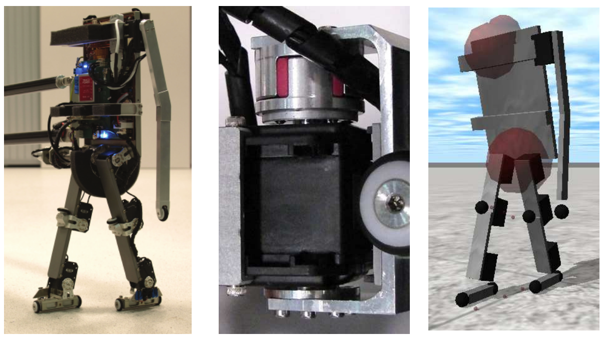

(a) LEO (b) Dynamixel RX-28 (c) Model of LEO

Fig. 1: LEO, one of the bipedal robots of the Delft BioRobotics Laboratory [1], [2]. In fig. 1b one of its joints is displayed. These joints are actuated by Dynamixel RX-28 motors which are connected to the robot via elastic couplings. In fig. 1c a model of LEO is displayed. With this model the learning process of algorithms can be simulated.

In section VII the performance and the mean time between failures (MTBF) of the algorithms are compared to each other while learning to swing up an inverted pendulum. The two most promising algorithms are tested on a physical inverted pendulum in section VII. In section VIII these two algorithms are tested on a simulation of LEO.

II. RELATED WORK

Reinforcement learning has been applied successfully to physical robots in previous research. In [3] policy gradient reinforcement learning is used to search for the optimal quadrupedal locomotion. Policy gradient reinforcement learning can assure smooth policies and can substantially increase the learning speed. To successfully apply this learning method, prior knowledge of the system is essential. The dependency on prior knowledge makes this method potentially less effective at adapting to new situations. In [4] actor-critic learning is applied. The random motions due to exploration are reduced by low-pass filtering with a discrete first order filter. Filtering at joint level is applied to protect the gearboxes from large differences in the torque signal. Although prior knowledge is not essential when applying actor-critic learning, it is added to get the necessary learning speed when learning on the physical robot.

We apply SARSA(λ) to quickly learn an optimal gait, online, on-policy and without adding prior knowledge. A disadvantage of learning with this method is the high frequent random motions it generates due to exploration and optimistic initialisation during initial learning. We aim to attenuate this disadvantage by adapting the learning algorithm.

III. REINFORCEMENT LEARNING

A reinforcement learning problem is defined by the tuple <S, A, P, R>. Where S is the set of possible states for the system to be in, A the set of possible actions the system can take, P the state transition distribution and R the reward function. The system tries to optimize the future reward which is defined as:

$$\sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Here γ defines how much rewards are discounted or in other words how much the system anticipates future rewards. The reward function and γ need to be specified. The system has to learn a policy π which takes the optimal action with respect to the future reward in every state. For this process to take place it is important that the state transitions are a constant distribution. Also the states in S must have the Markov property, meaning that states must have all the information in them to predict the future. In the case of LEO the system learns to apply the optimal torque, for each joint, in each state to get the optimal walking gait. These torques are learned with the SARSA(λ) algorithm. This algorithm tries to learn the optimal state-action values. The Bellman optimality equation for optimal state-action values is:

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a') \right]$$

Where the $\mathcal{P}^a_{ss'}$ comes from the state distribution and $\mathcal{R}^a_{ss'}$ from the reward function.

IV. FAILURE PREDICTION

We need to predict the MTBF of a physical system given a certain learning algorithm, in order to successfully adapt the learning algorithms. In this paper we take LEO as an example system on which we will compare five algorithms. Mechanical failure of LEO is usually caused by a gearbox breaking down [2]. The last gear in the gearboxes of LEO is usually the cause of failure. The material of this gear is able to withstand a number of stress cycles before it fails due to fatigue. A method to link complex combinations of cycles to fatigue failure is described by [5]. In this section the following procedure is used to predict the MTBF. First we determine the *stress cycles* that contribute most to fatigue failure. Second we approximate the *maximum torque* on the gear last gear during these cycles. Third we calculate the *maximum stress* in the last gear due to this torque. Fourth we approximate the number of times this stress cycle can be withstood. Finally we predict the MTBF based on the *predicted failure*.

Stress cycles: The condition in which cycles occur is defined by looking at the conditions in which the gearboxes actually fail. LEO is able to walk for more than 8 hours without failing while walking with a pre-programmed controller. However, the gearboxes of LEO fail after 5 minutes while performing random motion due to learning. Apart from falling, the impact of which is minimized by foam padding, the difference between the two situations is the number of backlash re-engagements (i.e. when the torque on the gear changes sign, the backlash is experienced and the gears re-engage). This leads to the assumption that the stress cycles during backlash re-engagement have the largest contribution to failure. We also assume that one backlash re-engagement leads to one stress cycle. The stress cycle can be expressed as a function of the torque while the gears re-engage.

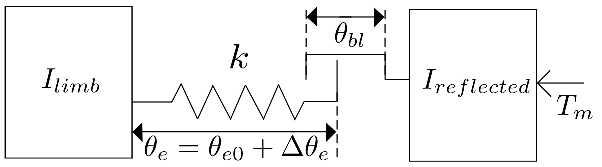


Fig. 2: A model of the backlash re-engagement assuming no friction. In this figure the rotations are represented as linear displacements. $\Delta\theta_e$ is the angular displacement in the elastic element, θ_{bl} is the backlash that is experienced, T_m is the torque at which the gears re-engage and $I_{reflected}$ is the reflected inertia of the motor.

Maximum torque: In order to predict the MTBF, the maximum torque on the last gear during a cycle needs to be calculated. In fig. 1b one of the motors used in LEO is depicted. The motor is connected to a limb via an elastic element. This elastic element is applied to protect the gearbox. The last gear of the gearbox is directly coupled to the elastic element. A simple model can be constructed by assuming zero friction. This model is visualized in fig. 2. The angle, θ_e, in this model is a relative angle between the motor and the joint. θ_{e0} is the equilibrium position of the elastic element and $\Delta\theta_e$ the angular displacement in the elastic element, θ_{bl} is the backlash between the gears, T_m is the torque supplied by the motor, $I_{reflected}$ is the reflected inertia of the motor and k is the stiffness of the elastic element. The maximum torque is calculated for the situation in which the elastic element is not displaced and backlash re-engagement occurs. This means that $I_{reflected}$ is accelerated by T_m over a distance of θ_{bl}, after which the elastic element is displaced and all energy supplied is stored in the elastic element. Assuming linear elasticity, the torque exerted by the elastic element is given by:

$$T_e = \Delta\theta_e \cdot k \tag{1}$$

The energy supplied to the system is the displacement times the motor torque. By assuming that the limb inertia is substantially larger than the reflected inertia of the motor this displacement in the elastic element can be calculated. The torque exerted by the elastic element is at its maximum when all the energy supplied is stored in the elastic element.

$$T_m \cdot (\Delta\theta_e + \theta_{bl}) = \frac{1}{2} \cdot \Delta\theta_e^2 \cdot k \tag{2}$$

By solving Equation 2 for $\Delta\theta$ and applying Equation 1 the maximum torque can be expressed;

$$T_{e_{max}} = T_m + \sqrt{T_m^2 + 2 \cdot T_m \cdot \theta_{bl} \cdot k} \tag{3}$$

Maximum stress: According to [6] the maximum stress in the tooth base of a spur gear is a function of the torque on this gear and a number of parameters which are based on the mechanics of the gearbox. The last gear of the gearbox is directly coupled to the elastic element therefore the last gear of the gearbox is subjected to $T_{e_{max}}$. The maximum stress in its tooth base is given by:

$$\sigma_{max} = \frac{2 \cdot T_{e_{max}}}{d_w \cdot b \cdot m_n} \cdot Y_{Fa} \cdot Y_{Sa} \cdot Y_{\epsilon} \cdot Y_{\beta} \tag{4}$$

Here the Y-parameters are correction factors to compensate for unmodelled effects and d_w represents the pitch circle , b the face width and m_n the module of the gear [6].

Number of cycles: In order to calculate the number of times this maximum stress can be experienced by the material, the stress cycle needs to be converted into an equivalent completely reversed cycle. This is a cycle which has its average stress at zero and has an equivalent number of times it can be withstood. According to [7] applying the Smith, Watson and Topper relationship creates accurate results. An equivalent completely reversed cycle can be calculated by using this relationship, assuming that the cycle has a stress amplitude, σ_a, of half the maximum stress:

$$\sigma_{ar} = \sqrt{\sigma_a \cdot \sigma_{max}} = \sqrt{\frac{\sigma_{max}}{2} \cdot \sigma_{max}} = \frac{1}{\sqrt{2}} \cdot \sigma_{max} \tag{5}$$

The relationship between a completely reversed cycle and the number of times it can be withstood before failure is approximated in [7] by:

$$N_f = \frac{1}{2} \cdot \left(\frac{\sigma_{ar}}{\sigma'_f} \right)^{\frac{1}{d}} \tag{6}$$

Here σ'_f and d are fitting parameters from [7].

Predicted failure: The last step of the procedure is the failure prediction. Failure of the gear can be predicted by summing the inverse of N_f for each backlash re-engagement, assuming that each of the 45 teeth of the gear are equally fatigued and looking at when this reaches the value of one.

$$J = \sum_{i=1}^{p} \frac{1}{45 \cdot N_{fi}} \tag{7}$$

The prediction of failure based on $J \geq 1$ is also assumed to predict the MTBF. This predicted MTBF ignores material fatigue due to taking steps and falling. The parameters used for this summation are given in table I. There are two ways the algorithm can influence the value of this summation. It can influence p, the number backlash re-engagements and it can influence T_m, the torque at which it re-engages (see Equation 3). We will investigate 5 algorithms that either influence p, T_m or both. In order to focus on the early learning phase, the MTBF is based on the first 1200 actions generated by the algorithm.

V. ALGORITHMS

The MTBF and learning performance of six different algorithms are studied. They are presented below.

TABLE I: The parameters used in equation 3, 4, 5, 6 and 7. In the first column are fitting parameters from [7] and model parameters. In the last two columns are parameters based on the mechanics of the gearbox [6].

θ_{bl}	0.01 $[rad]$	b	4 $[mm]$	Y_{Fa}	2.15
k	106 $[Nm \cdot rad^{-1}]$	m_n	0.4 $[mm]$	Y_{Sa}	2
d	-0.262	d_w	18.5 $[mm]$	Y_{ϵ}	1.22
σ'_f	4402 $[MPa]$			Y_{β}	1

SARSA(λ) : SARSA(λ) algorithm is used as a benchmark for the other algorithms. Tile coding is applied in order to speed up learning. Tile coding can increase the learning speed of the process because it helps to generalize state-action values [8]. The SARSA(λ) uses ϵ-greedy exploration to explore state-actions. The other algorithms are variations of this algorithm and they also use tile coding, ϵ-greedy exploration and the same learning parameters.

Low-pass filter: The torques generated by the SARSA(λ) algorithm can be low-pass filtered before being applied to the robot [4]. This low-pass filter can be implemented with a discrete first order filter, given by:

$$a_{filtered_k} = \alpha \cdot a_k + (1 - \alpha) \cdot a_{filtered_{k-1}} \tag{8}$$

This filter transforms a discrete action signal into a continuous action signal. A first order filter is used to keep the dynamics of the filter as simple as possible. If α approaches zero, the predicted MTBF approaches infinity. This low-pass filter algorithm with an α less than one will lose its Markov property because the previous action is not in the state.

Markov low-pass filter: The third algorithm adds the previous action to the state thus preserving the Markov property at the cost of increasing the number of state dimensions.

Integrating controller: Integrating a signal via a discrete integrator can result in a signal with less high frequencies. In [9] the controller is integrated to accurately regulate a system without having a large number of actions. We will use an integrating controller that closely resembles a delta demodulator to attenuate high frequencies [10]. The SARSA(λ) algorithm learns to increase, decrease or maintain the current torque instead of letting it decide which torque to apply. The integrated signal has a fixed value, Δ, at which it can change. This value is made dimensionless by defining its size relative to the size of the action space of the corresponding action. When this Δ approaches zero the predicted MTBF approaches infinity. The system will lose its Markov Property because the previous action is not in the state.

Markov integrating controller: The fifth algorithm adds the previous action to the state, thus preserving the Markov property at the cost of increasing the number of state dimensions.

PADA: So far each algorithm that increases the MTBF either loses the Markov property or increases the number of state dimensions. The Markov property can be preserved without adding a state dimension by implementing the integrating controller differently. SARSA(λ) uses state-action values to learn an optimal policy. An important observation is that while working with state-action values, action selection can be a function of the previous action. The previous action dependent actions algorithm, or PADA algorithm, uses this fact. When the next action is a function of the current action the Bellman optimality equation becomes:

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in f(a)} Q^*(s',a') \right]$$

With $f(a)$ a function that limits the applicable actions. This will change the value of the optimal state-action but it is still computable. The dependency of the next action to the current action is equivalent to the current action depending on the previous action:

$$a' \in f(a) \mapsto a_{t+1} \in f(a_t) \mapsto a_t \in f(a_{t-1})$$

The integrating controller can be implemented by only evaluating the actions that are on a fixed distance from the previous action and evaluating the previous action itself:

$$f(a_{t-1}) = \{a | \ a - a_{t-1} \in \{\pm\Delta, 0\}\} \tag{9}$$

From these three state-action values the highest is selected. While learning motions, the state-action values learned while using this PADA algorithm are approximately equal to the state-action values learned by a SARSA(λ) algorithm. Switching between the PADA algorithm and the SARSA(λ) can be done by starting to evaluate every action instead of three ($f(a) = A$).

VI. LEARNING PERFORMANCE

Increasing the MTBF will cause the learning performance to change. The actual MTBF is roughly estimated by Equation 7 therefore it is difficult to optimise between increasing it and decreasing learning performance. The objective is therefore defined as the 95% confidence that the average performance is less than 33% decreased, while increasing the MTBF.

The learning performance of the algorithms is characterized by two averages. These are the average end performance and time constant.

End performance: The performance during the learning process is computed by summing up all rewards given during a test run. In this run all actions are based on the greedy policy. The end performance is calculated by looking at the average performance of the last 10% test runs of a trial.

Time constant: The increase of performance during learning can be approximated by an exponent which asymptotically goes to the end performance. The rate at which it converges is defined by $1 - e^{-1/\tau}$ where τ is the time constant. The time constant of a trial is computed by looking at when $1 - 1/e^2 = 0.865 = 86.5\%$ of the end performance is reached and dividing this by two. Averaging the time constants of the trials gives the average time constant.

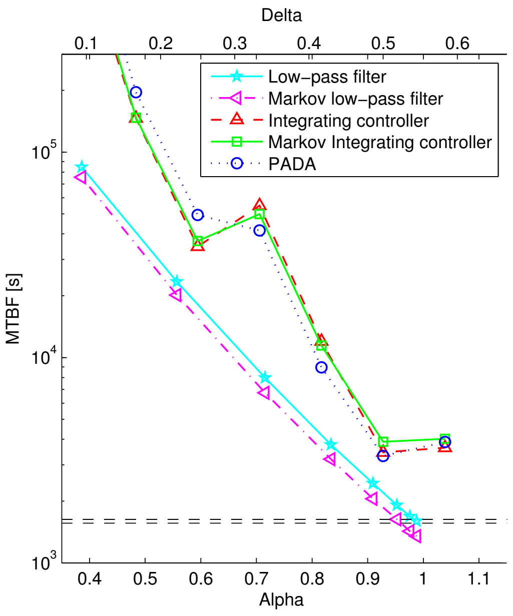


Fig. 3: The MTBF can be increased by decreasing the Δ or α parameter. The horizontal dashed lines represent the confidence interval of the average MTBF of the SARSA(λ) algorithm.

VII. INVERTED PENDULUM SIMULATION

The MTBF and learning performance can be investigated by simulating how each of the algorithms learns to swing up an inverted pendulum to its unstable equilibrium. The pendulum consists of a point mass, of 2 [kg], which is located 0.2 [m] from the joint and its moment of inertia is 0,048 $[kg \cdot m^2]$. The torque on the pendulum is between ± 1.5 $[Nm]$ and assumed to be generated by the same hardware as in LEO. The state of this system is sampled at 20 [Hz] and it consists of the angle θ and the speed $\dot{\theta}$ of the pendulum. Its reward function is $-5\theta^2 - 0.1\dot{\theta}^2 - a^2$, with a the applied torque on the pendulum. Each of the algorithms will learn three actions. Their averages are computed over 29 trials.

SARSA(λ): The SARSA(λ) algorithm learns to put ± 1.5 or 0 $[Nm]$ torque on the joint to get the angle θ to zero. The algorithm has a learning rate (α) of 0.2, a discount rate (γ) of 0.99, an exploration rate (ϵ) of 0.05, and a trace decay rate (λ) of 0.92. These parameters have not been tuned for any specific system. They will be used by all six algorithms. The average end performance of the SARSA(λ) algorithm is -851. This average has a 95% confidence interval of 39. 33% of -851 is approximately -281 therefore the average end performance of other algorithms should be above -851-281+39=-1093 with a confidence of 95%. The average time constant is 393 $[s]$. This average has a confidence interval of 26 $[s]$, therefore the average time constants of the other algorithms should be below 498 $[s]$. The MTBF of this algorithm is 1596 $[s]$ with a confidence interval of 34 $[s]$.

Other algorithms: In fig. 3 the MTBF of the other algorithms are displayed as a function of either the α or Δ respectively. All algorithms are able to significantly increase the MTBF and have small confidence intervals. In fig. 4 the end performance of the algorithms as a function of

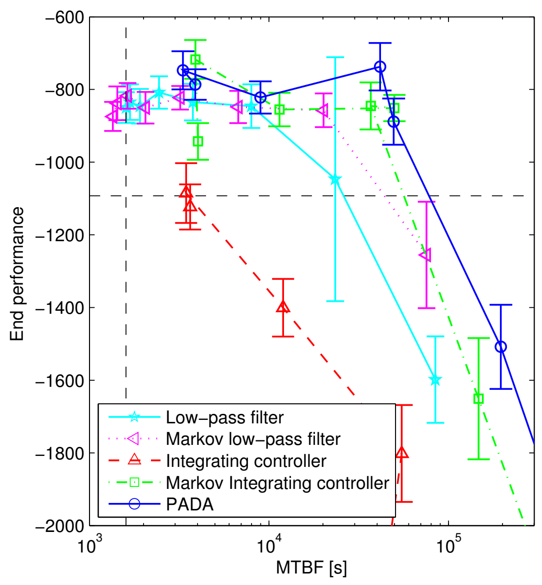


Fig. 4: Increasing the MTBF generally has a decreasing effect on the end performance. The integrating controller algorithm is unable to achieve an equivalent end performance. The horizontal dashed line is the minimal end performance that should be reached with a 95 % certainty. The vertical dashed line is the MTBF of the SARSA(λ) algorithm.

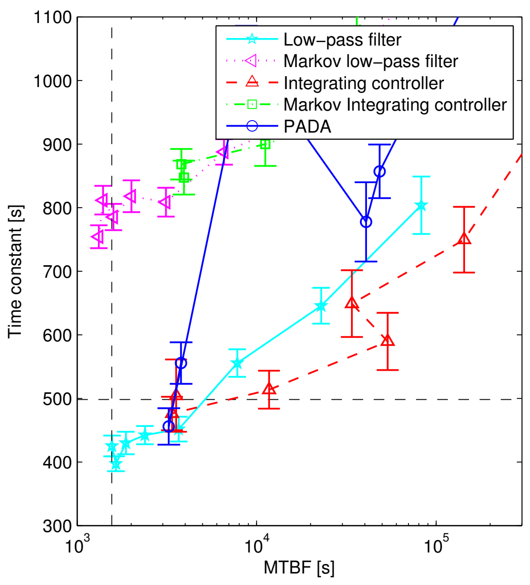


Fig. 5: Increasing the MTBF generally has a increasing effect on the time constant. Both the algorithms that increased the number of state dimensions to preserve the Markov property are unable to achieve equivalent learning speed. The horizontal dashed line is the maximal time constant that should be reached with a 95 % certainty. The vertical dashed line is the MTBF of the SARSA(λ) algorithm.

the MTBF is visualized. The plain integrating controller algorithm is the only one that is unable to achieve equivalent end performance. As shown in fig. 5, the time constant generally increases when the MTBF is increased. Both algorithms that increased the number of state dimensions are unable to achieve equivalent time constants. This leaves two algorithms which are able to reach the objective. The low-pass filter algorithm with an α of 0.834 or larger and the PADA algorithm with a Δ of 0.5. Both these algorithms are able to increase the MTBF by a factor of two. The PADA algorithm with this delta increases the MTBF by reducing the number of backlash re-engagements during exploration while the low-pass filter decreases the torque at which it re-engages.

INVERTED PENDULUM EXPERIMENT

The PADA algorithm with a Δ of 0.5, the low-pass filter algorithm with an α of 0.834 and a SARSA(λ) are applied to a physical inverted pendulum. All algorithms use the same learning parameters. The action supplied to this pendulum is the voltage on the motor rather than the torque delivered by the motor. This voltage is between ± 3 $[V]$. The physical properties of this system are given in [11]. Assuming a linear relationship between voltage and torque, and that it is generated by the same hardware as in LEO, the MTBF can be calculated. The averages are computed over 28 trials.

MTBF: The SARSA(λ) algorithm has a MTBF of $6.92 \cdot 10^7$ $[s]$, the low-pass filter algorithm a MTBF of $1.31 \cdot 10^8$ $[s]$ and the PADA algorithm a MTBF of $1.39 \cdot 10^8$ $[s]$. The PADA algorithm and the low-pass filter algorithm are both able to increase the MTBF by a factor of two.

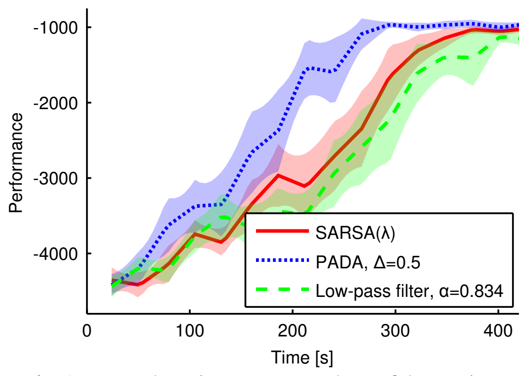


Fig. 6: Average learning curve and confidence interval of the physical inverted pendulum.

learning performance: In fig. 6 the average performance as a function of the number of steps is visualized. All algorithms have equivalent end performance. From this figure it is clear that the PADA algorithm learns significantly faster than the other two. This indicates that there are cases in which the PADA algorithm does not have to sacrifice performance in order to increase the MTBF.

VIII. LEO SIMULATION

The learning process on LEO can be simulated to assess the properties of both algorithms on complex tasks relative to SARSA(λ). The model used to simulate LEO is visualized in fig. 1c. It has 10 state dimensions and is highly non-linear. The algorithm properties are computed by averaging over 15 trials. The SARSA(λ) algorithm learns to pick the optimal voltage, out of the set $\{-1,\ -2/3,\ -1/3,\ 0,\ 1/3,\ 2/3,\ 1\} \cdot 10.7$ $[V]$, to get the optimal walking gait. It learns this for

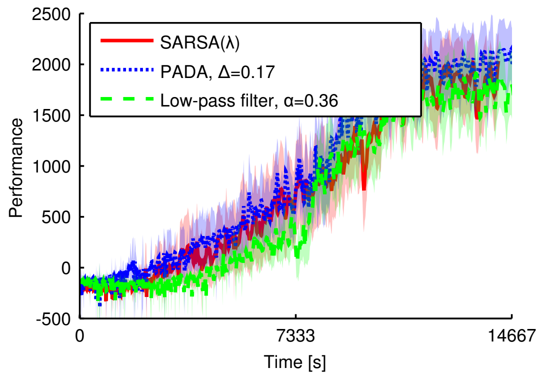


Fig. 7: Average learning curve of the simulated learning process of LEO. The SARSA(λ) learning curve is displayed as a reference. The other two methods have a MTBF of 39,000 $[s]$.

both the two hips and the swing leg knee joint therefore the number of action dimensions is three.

Small Δ or α values will cause the learning performance to drop, making them undesirable. The predicted MTBF of an algorithm is assumed to be large in the exploration phase relative to the optimization phase. In order to apply these small values without a significant drop in performance they could be used during the exploration phase only. After the exploration phase (7,333 $[s]$ in LEO's case) the algorithms can be switched back to SARSA(λ). In most trials the robot has already found a sub-optimal walking gait at this point. The SARSA(λ) algorithm optimizes this gait for another 7,333 $[s]$. Cutting up the learning process into parts changes the apparent system dynamics (which may require some relearning) but it has advantages when looking at the MTBF. In nature many learning processes are also made out of parts. For instance rat foetuses already start to learn motions while they are still in their mothers womb [12].

SARSA(λ): The SARSA(λ) algorithm has an average end performance of 1760 and a confidence interval of 262. The average end performance of the other algorithms should be be above 1441. The algorithm has a MTBF of 362 $[s]$.

Other algorithms: As can be seen in fig. 8, the MTBF can be increased further when both algorithms are switched back after 7,333 $[s]$. In fig. 7 the learning curves of a PADA and a low-pass filter which both have the same MTBF are displayed. These two algorithms both increase the MTBF by a factor of 108 to approximately 39,000 $[s]$. Also the SARSA(λ) algorithm is displayed as a reference. While the PADA algorithm is able to learn with small values of Δ and switch back to SARSA(λ) without losing performance, the low-pass filter is not. Smaller values of Δ might also be possible when applying the PADA algorithm. The PADA algorithm is clearly better at keeping the learning performance equivalent while increasing the MTBF.

IX. CONCLUSIONS

The PADA and low-pass algorithm are both able to increase the MTBF considerably while keeping the learning performance equivalent to that of SARSA(λ). The PADA algorithm does not violate the Markov property during learning. The low-pass filter algorithm does, which is a disadvantage. The PADA algorithm has more potential to increase the MTBF than the low-pass filter when applied to more complex systems. The PADA algorithm can increase the MTBF by a factor of 108 in case of LEO. In some cases it can even out-perform the SARSA(λ) algorithm.

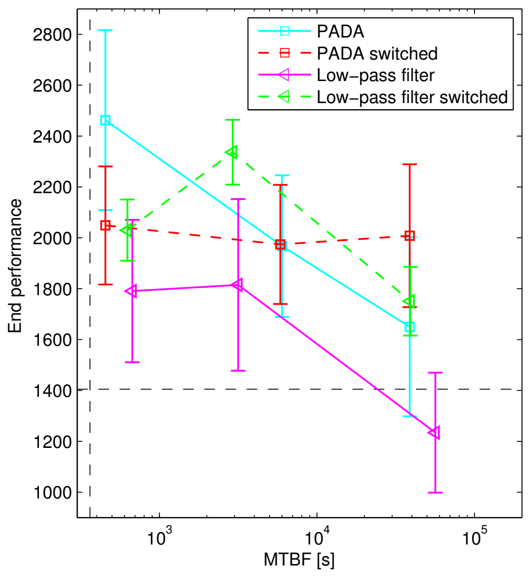


Fig. 8: Increasing the MTBF generally has a decreasing effect on the end performance. This effect can be mitigated by switching back to SARSA(λ) after the exploration phase.

REFERENCES

[1] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker, "The design of leo: a 2d bipedal walking robot for online autonomous reinforcement learning," *International Conference on Intelligent Robots and Systems*, pp. 3238 – 3243, 2010.

[2] E. Schuitema, "Reinforcement learning on autonomous humanoid robots," Ph.D. dissertation, Delft University of Technology, 2012.

[3] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," *ICRA*, pp. 2619–2624, 2004.

[4] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, pp. 283–302, 1997.

[5] N. Dowling, "Fatigue failure predictions for complicated stress-strain histories," *Illinois Univ., Urbana. Jan. 1971. 87*, 1971.

[6] D. Muhs, H. Wittel, M. Becker, D. Jannasch, and J. Voiek, *Roloff/Matek Machine-onderdelen*, R. Heyer, Ed. Academic Service, 2005.

[7] N. E. Dowling, "Mean stress effects in stress-life and strain-life fatigue," *Society of Automotive Engineers*, 2004.

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

[9] R. Hafner and M. Riedmiller, "Neural reinforcement learning controllers for a real robot application," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2098–2103.

[10] R. Steele, "Srn formula for linear delta modulation with ban-limited flat and rc-shaped gaussian signals," *Transactions on Communication*, vol. 28, pp. 1977–1984, 1980.

[11] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema, "Efficient model learning methods for actor-critic control," *TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, vol. VOL. 42, pp. 591–602, 2012.

[12] G. A. Kleven, M. S. Lane, and S. R. Robinson, "Development of interlimb movement synchrony in the rat fetus." *Behavioral neuroscience*, vol. 118, no. 4, p. 835, 2004.

Bibliography

[1] E. Schuitema, M. Wisse, T. Ramakers, and P. Jonker, "The design of leo: a 2d bipedal walking robot for online autonomous reinforcement learning," *International Conference on Intelligent Robots and Systems*, pp. 3238 – 3243, 2010.

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

[3] E. Schuitema, *Reinforcement Learning for autonomous humanoid robots.* PhD thesis, Technische Universiteit Delft, 2012.

[4] N. E. Dowling, "Mean stress effects in stress-life and strain-life fatigue," *Society of Automotive Engineers*, 2004.

[5] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," *ICRA*, pp. 2619–2624, 2004.

[6] B. van Vliet, W. Caarls, E. Schuitema, and P. Jonker, "Accelerating reinforcement learning on a robot by using subgoals in a hierarchical framework,"

[7] N. Dowling, "Fatigue failure predictions for complicated stress-strain histories," *Illinois Univ., Urbana. Jan. 1971. 87*, 1971.

[8] D. Muhs, H. Wittel, M. Becker, D. Jannasch, and J. Vossiek, *Roloff/Matek Machineonderdelen.* Academic Service, 2005.

[9] J. J. Craig, "Introduction to robotics: mechanics and control," 2004.

[10] R. Steele, "Srn formula for linear delta modulation with ban-limited flat and rc-shaped gaussian signals," *Transactions on Communication*, vol. 28, pp. 1977–1984, 1980.

[11] R. Hafner and M. Riedmiller, "Neural reinforcement learning controllers for a real robot application," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2098–2103, IEEE, 2007.

[12] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema, "Efficient model learning methods for actor-critic control," *TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, vol. VOL. 42, pp. 591–602, 2012.

[13] G. A. Kleven, M. S. Lane, and S. R. Robinson, "Development of interlimb movement synchrony in the rat fetus.," *Behavioral neuroscience*, vol. 118, no. 4, p. 835, 2004.