2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

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 five 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). 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 and they need to be able to withstand these stresses in order to prevent failure.

The bipedal robot LEO (see Fig. 1a) was designed to withstand large stresses while learning 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 possible 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 mean time between failures (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. In Section VII the performance and the 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.

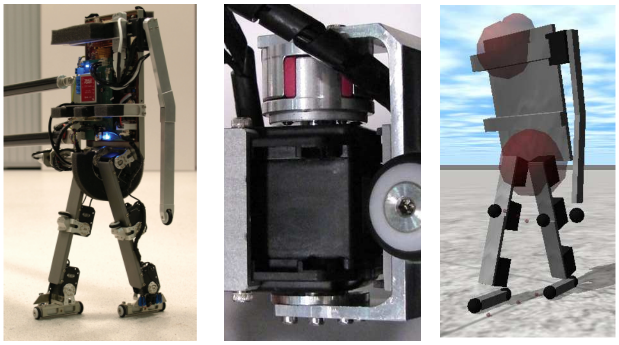
(a) LEO (b) Dynamixel RX-28 (c) Model of LEO

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.

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 to quickly learn an optimal gait, on-line, 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

978-1-4673-6357-0/13/$31.00 ©2013 IEEE

initialisation during initial learning. We aim to attenuate this disadvantage by various methods of filtering the actions.

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. For each time-step *t*, the system tries to optimise the discounted sum of future rewards, r_{t+k}:

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

Here γ defines how much rewards are discounted. 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 transition distribution does not change. 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 voltage, for each joint, in each state to get the optimal walking gait. These voltages are learned with the SARSA algorithm which tries to learn optimal state-action values. The Bellman optimality equation for optimal state-action values is:

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

Where s' is the next state, a' the next action, $P^a_{ss'}$ comes from the state transition distribution and $R^a_{ss'}$ from the reward function. The algorithm stores an approximation of the state-action values and updates these, each time-step, with:

$$Q(s,a) = Q(s,a) + \alpha \left[r_{t+1} + \gamma Q(s',a') - Q(s,a) \right]$$

Here α defines the learning rate of the process.

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 predict the MTBF of LEO for different action filtering 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 eight hours without failing while walking with a pre-programmed controller. However, the gearboxes of LEO fail after five 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.

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 Eq. 2 for $\Delta\theta$ and applying Eq. 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

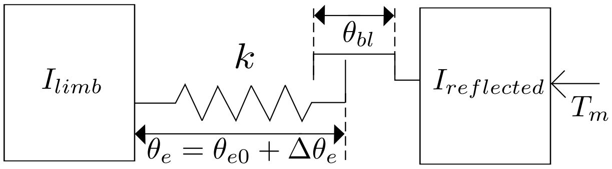

Fig. 2: A model of the backlash re-engagement assuming no friction. In this figure the rotations are represented as linear displacements. $\Delta\theta_e$ is the angular displacement in the elastic element, θ_{bl} is the backlash that is experienced, T_m is the torque at which the gears re-engage and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

of the gearbox is subjected to $T_{e_{max}}$. The maximum stress in its tooth base is given by:

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

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:

$$\sigma_{ar} = \sqrt{\sigma_a \cdot \sigma_{max}} = \sqrt{\frac{\sigma_{max}}{2} \cdot \sigma_{max}} = \frac{1}{\sqrt{2}} \cdot \sigma_{max} \qquad (5)$$

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

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

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.

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

Averaging the prediction of failure of multiple trials results in the predicted MTBF. This predicted MTBF ignores material fatigue due to taking steps and falling. The parameters used for this summation are given in table I.

An action set with only the maximum and minimum action (which is often used) would be the worst-case scenario and produces the smallest MTBF. The MTBF increases and converges as the number of actions increase. Given a certain action set, there are two ways the algorithm can influence the MTBF. It can influence p, the number of backlash re-engagements and it can influence T_m, the torque at which it re-engages (see Eq. 3). We will investigate five 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.

TABLE I: The parameters used in Eqs. 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

V. ALGORITHMS

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

SARSA : The SARSA 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 algorithm 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 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:

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

This filter transforms a discrete action signal, a, into a continuous action signal, $a_{filtered}$. 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 filtered action, $a_{filtered_{k-1}}$, is not in the state.

Markov low-pass filter: The third algorithm adds this 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 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 integrated action is not in the state.

Markov integrating controller: The fifth algorithm adds the integrated 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 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:

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

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:

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

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 algorithm. Switching between the PADA algorithm and the SARSA can be done by starting to evaluate every action instead of three ($f(a) = A$).

VI. LEARNING PERFORMANCE

Increasing the MTBF causes the learning performance to change. The actual MTBF is roughly estimated by Eq. 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 still increasing the MTBF.

The learning performance of the algorithms is characterized by two averages. These are the average end performance and rise time. An example of these are given in Fig. 6.

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.

Rise time: The rise time is computed by using the relative performance difference between the first test run and the end performance. It is computed by looking at the time taken to rise to 95% of its relative end performance. Averaging the rise times of the trials gives the average rise time.

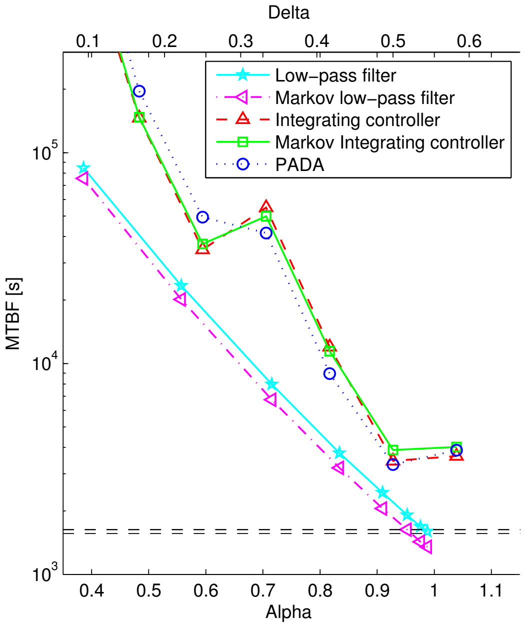

Fig. 3: The MTBF can be increased by decreasing the Δ or α parameter. The horizontal dashed lines represent the confidence interval of the average MTBF of the SARSA(λ) algorithm.

VII. INVERTED PENDULUM SIMULATION

We start to make an initial selection of promising algorithms by comparing the different algorithms on a simulated inverted pendulum. The pendulum consists of a point mass, of 2 [kg], which is located 0.2 [m] from the joint, 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, with respect to its unstable equilibrium and the velocity, $\dot{\theta}$, 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 θ to zero (almost worst-case). 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 and 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 rise time is 681 $[s]$. This average has a confidence interval of 67 $[s]$, therefore the average rise time of the other algorithms should be below 839 $[s]$. 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 the MTBF

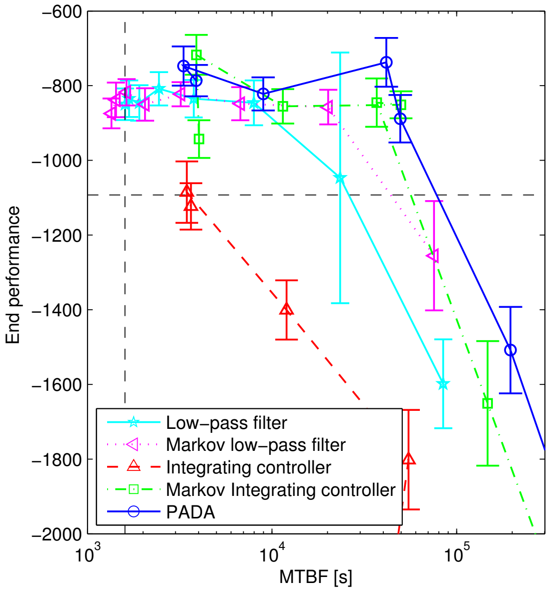

Fig. 4: Increasing the MTBF generally has a decreasing effect on the end performance. The end performance of the algorithm should be above the horizontal dashed line with a 95% certainty. The integrating controller algorithm is unable to achieve an equivalent end performance. The vertical dashed line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

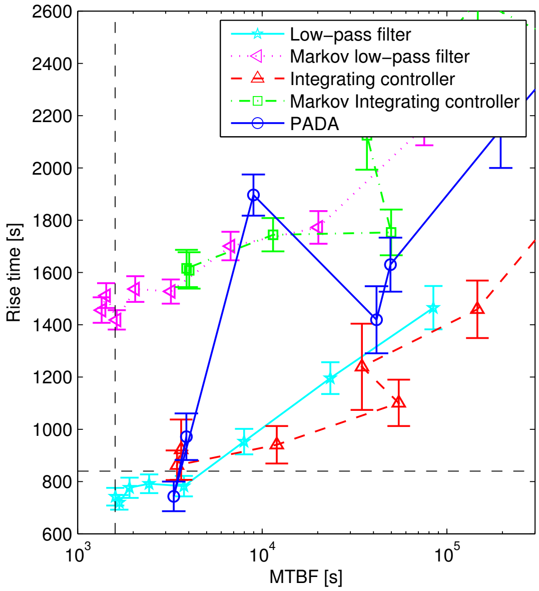

Fig. 5: Increasing the MTBF generally has a increasing effect on the rise time. The rise time of the algorithm should be below the horizontal dashed line with a 95% certainty. Both the algorithms that increase the number of state dimensions to preserve the Markov property are unable to achieve equivalent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

is visualized. Only the plain integrating controller algorithm is unable to achieve equivalent end performance. As shown in Fig. 5, the rise time generally increases when the MTBF is increased. Both algorithms that increased the number of state dimensions are unable to achieve equivalent rise times. 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) algorithm are applied to a physical inverted pendulum. All algorithms use the same learning parameters. The action supplied to this pendulum is a motor voltage between ± 3 $[V]$. The pendulum is sampled at 33 $[Hz]$ and its physical properties 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.

Learning performance: In Fig. 6 the average performance as a function of time is visualized. All algorithms have

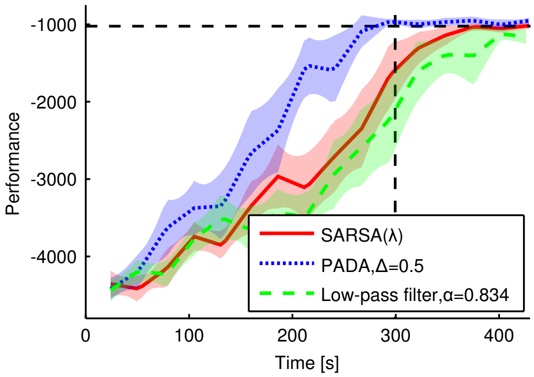

Fig. 6: Average learning curves and confidence interval of the physical inverted pendulum. The horizontal line indicates the end performance, and the vertical line indicates the rise time, of the SARSA(λ) algorithm.

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 and uses the ODE engine. The simulated LEO is rewarded for forward motion and is sampled at 30 $[Hz]$. 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,

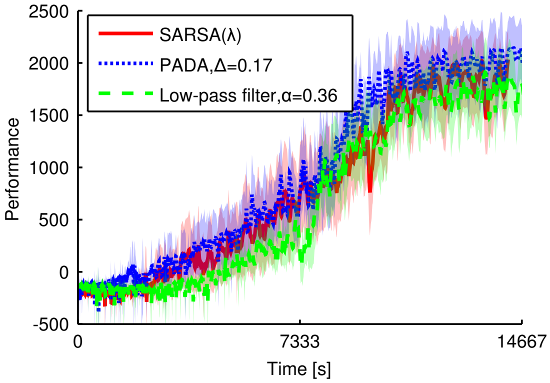

Fig. 7: Average learning curve of the simulated learning process of LEO. The SARSA(λ) learning curve is displayed as a reference. The other two methods have a MTBF of 39,000 $[s]$.

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 predicted MTBF of an algorithm is assumed to be large in the exploration phase relative to the optimization phase. In 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 (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 changes the apparent system dynamics (which may require some relearning) but it has the previously described advantage. 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 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.

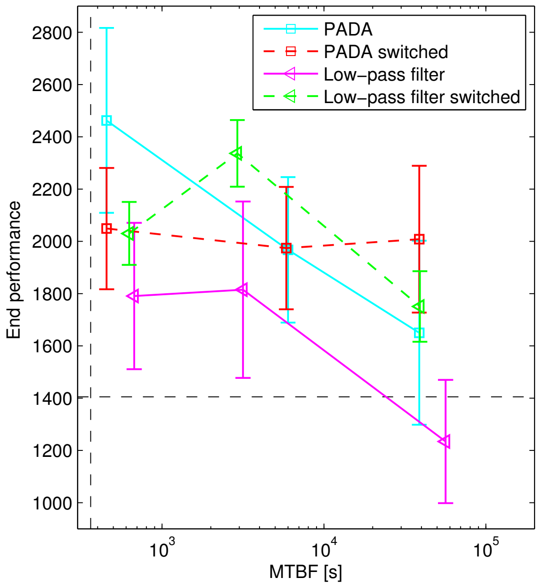

Fig. 8: Increasing the MTBF generally has a decreasing effect on the end performance. This effect can be mitigated by switching back to SARSA(λ) after the exploration phase.

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," in *Proc. IROS*, 2010, pp. 3238 – 3243.

[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," in *Proc. ICRA*, 2004, pp. 2619–2624.

[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. E. Dowling, "Fatigue failure predictions for complicated stress-strain histories," Illinois Univ. at Urbana, Tech. Rep. AD0736583, 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," in *Proc. SEA fatigue congress*, 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 *Proc. ICRA*, 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, part B*, 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.