Learning optimal gait parameters using the episodic Natural Actor-Critic method

G.P.A. Knobel BSc

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Learning optimal gait parameters using the episodic Natural Actor-Critic method

Master of Science Thesis

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

G.P.A. Knobel BSc

March 27, 2011

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Learning optimal gait parameters using the episodic Natural Actor-Critic method

by

G.P.A. Knobel BSc

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: March 27, 2011

Supervisor(s):

prof.dr. R. Babuška

dr. G.A. Delgado Lopes

Reader(s):

prof.dr.ir. P.P. Jonker

ir. I. Grondman

Abstract

Wheeled robots (and everyday vehicles) move very well on smooth surfaces such as rails or roads, but perform poorly on rough surfaces, which are estimated to cover more than 99% of the land area of earth. On rough terrain legged robots have the advantage over their wheeled counterparts. However, the increased versatility of legged robots raises extra challenges. One of most important is finding motion controllers that yield efficient locomotion in terms of achievable velocity or lower power consumption.

The mechanical complexity of legged platforms together with the difficulty in modelling the intermediate interactions with the ground suggest the use of learning techniques for finding optimal gait parameters. In this thesis the gait is generated using the Switching Max-Plus-linear model, a recently proposed gait generation method that offers intuitive modelling and can ensure stability of the robot under changing gait parameters. This method operates by enforcing the synchronization of multiple discrete event circuits, each composed of two events: leg lift-off and touchdown.

Learning is accomplished using the episodic Natural Actor-Critic (eNAC) method, which is a Reinforcement Learning (RL) technique. RL is a learning framework inspired by the way animals learn to deal with new situations. The most important properties of eNAC are that it does not require a model of the system and is capable of handling large, continuous state spaces such as encountered in legged robotics, making it particularly suitable for the problem of optimal gait learning. The actions applied to the system are the lift-off and touchdown angles that define, together with the timings of the events, the reference trajectories for the legs to follow. A model of a hexapod robot has been developed and is used to perform the learning process.

Results indicate that the eNAC method can successfully be applied to the problem of learning optimal gait parameters, where on average an increase of 20% was found in velocity, with a maximum increase of 120%.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

Ever since I was a little boy I have been fascinated by technology: how do things work, what principles are behind it and especially robotics. Conscious and unconscious the decisions I have made during my career as a student led me to this very place, in which I was able to do research in the field of my childhood interests. Although circumstances forced me to take some other directions than foreseen at the start of my research, it has been a very educational experience.

I would like to thank prof.dr. R. Babuška, whose enthusiastic and clear way of teaching was one of the main reasons for me to specialize in Systems and Control. Also, I would like to thank my daily supervisor dr. G.A. Delgado Lopes, who sparked my interest in the field of legged robotics at the very start of my master.

Furthermore, I would like to thank my parents, brothers and friends, who, despite the fact that they have no technological knowledge, listened to my problems, tried to come up with useful suggestions and supported me when necessary.

Delft, University of Technology — G.P.A. Knobel BSc
March 27, 2011

Chapter 1

Introduction

1-1 Mobility in robotics

Vehicles and robots equipped with wheels perform very well on relatively flat, smooth terrain such as rails or roads. However, they perform poorly on rough terrain, such as e.g. rocky terrain, loose sand, mud, etc, which are estimated to cover more than 99% of the land area of earth. However, if we take a look at the animal kingdom we see that during many millions of years of evolution the majority of land animals developed legs, not wheels. With these legs animals are capable of manoeuvring through smooth as well as rough terrain. Robots equipped with legs have thus the potential to cover a far larger part of the earth compared to robots equipped with wheels. This makes it possible to use them for instance for urban search and rescue missions, in which the terrain is often full of debris.

A very recent catastrophe where legged robots could have been employed is the aftermath of the earthquake and the thereby triggered tsunami that struck Japan on March 11, 2011. Due to these events several nuclear facilities were damaged and radioactive material leaked, making it dangerous for humans to come close to the reactor to inspect and repair the damage. Legged robots, on the other hand, would not be harmed (or to a less extent than humans) by the radiation and are in theory capable of manoeuvring through the terrain. This raises the question: why do we not equip robots with legs?[1]

One of the main challenges of legged robotics is to find correct gait parameters that will yield an efficient walk in terms of the velocity and power consumption, which is the problem that will be addressed in this thesis.

In this section a short overview of the history of legged robotics is given, along with a method of measuring the efficiency of a gait. Furthermore, an overview of gait generation and optimization methods previously applied to legged robots is shown.

[1] At http://spectrum.ieee.org/automaton/robotics/industrial-robots/...
...japan-robots-to-fix-troubled-nuclear-reactors an interesting article can be found why current robots cannot be employed in these kinds of environments.

History Legged locomotion was already studied as early as the 1870s by e.g. Eadweard Muybridge who documented the walking and running behaviour of over 40 mammals, including humans. One of the first serious attempts of legged locomotion in robotics was done in the 1960s by Robert McGhee[2]. He used a computer controlled hexapod robot, that could walk with a number of standard gaits, turn, walk sideways and overcome simple obstacles. Other major developments were done by Marc Raibert in the 1980s. He founded the Leg Laboratory at Carnegie-Mellon University, in which some of the earliest running and dynamically balancing robots were developed, such as a planar one-legged machine that hops in place, travels at a specified velocity, keeps its balance when disturbed and jumps over small obstacles.

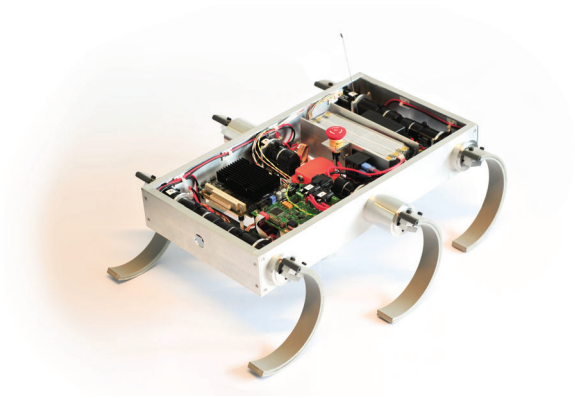


Figure 1-1: This figure shows Zebro, the "**Zes**benige **ro**bot". The morphology of the robot is inspired by that of insects, in particular how cockroaches move.

From then on the development of legged robotics started growing and this has led to some well known (commercially available) legged robots, such as Honda's ASIMO humanoid robot and Sony's AIBO robot. Furthermore, large contributions in the field of legged robots have been delivered by Boston Dynamics[3]. This company is responsible for some of the most advanced legged robots to date, such as BigDog and LittleDog. However, one of the most important breakthroughs in recent years came with the development of RHex by a consortium of universities around 2000, funded by DARPA (part of the United States Department of Defense). RHex is inspired by the morphology of insects, in particular on how cockroaches move. Cockroaches have six legs, three on each side of the body. They move their legs in such a way that the supporting legs always form a triangle, which gives great stability. Applying this to RHex allows it to walk with speeds over one body length per second, even on rough surfaces. In this thesis use is made of Zebro (Figure 1-1); a robot built along the same principles as RHex. Zebro is developed at the Delft Center for Systems and Control (DCSC), part of the faculty of Mechanical, Maritime and Materials Engineering (3mE) of Delft University of Technology (TU Delft).

[2] http://spectrum.ieee.org/robotics/robotics-software/march-of-the-sandbots/
[3] http://www.bostondynamics.com/bd_about.html

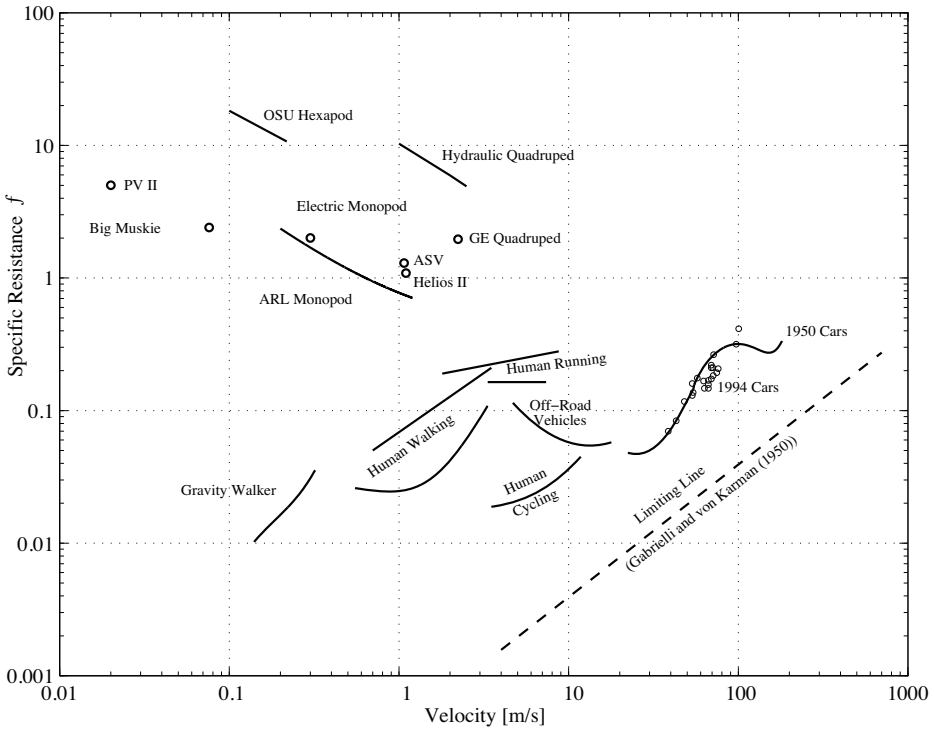


Figure 1-2: A comparison of specific resistances of robots, cars and humans. The figure is adopted from Gregorio et al. (1997).

The cost of locomotion A large amount of research has been performed in the field of the cost of locomotion in animals. E.g. Nishii (2000) concluded that the gait used by legged animals is highly optimized with respect to the energetic cost, based on tests with a simulation model of a hexapod robot. Before that, it was concluded by Hoyt and Taylor (1981) that for horses it holds that the natural gait uses the lowest energy cost possible at that speed. This was discovered by measuring the oxygen consumption during the execution of several gaits.

One way of expressing the relation between power consumption and velocity is by the use of the specific resistance, which is given by (Gabrielli and Von Karman, 1950)

$$f_{\text{sr}} = \frac{P}{mg\dot{x}} \tag{1-1}$$

where P is the power consumption of the robot, m the mass of the robot, g denotes the gravitational acceleration and $\dot{x}$ denotes the velocity of the robot in forward direction.

Due to the fact that the specific resistance is a dimensionless quantity, as follows from Eq. (1-1), it can be used to compare different types of moving objects, such as robots, cars and animals, to each other (Figure 1-2). An example of a recent application of the specific resistance in combination with legged robots can be found in Weingarten et al. (2004), where the gait of a hexapod robot is optimized.

Gait generation Different methods of gait generation have been studied in literature (Holmes et al., 2006). They range from methods that rely heavily on exact control of the foot positions, such as the Zero-Moment Point (ZMP) method, to methods that do not have any control at all, such as the compass gait that relies purely on gravity. However, these methods are primarily applied to bipedal robots. In between there is a class of gait generation methods that only need moderate control, e.g. the Spring-Loaded Inverted Pendulum (SLIP) method, which models a robot as a point mass attached to a spring and can be applied to multi-pedal robots. SLIP has been proposed as a template model for the sagittal dynamics of most animals. Furthermore, for multi-pedal robots Central Pattern Generators (CPGs) can be used (e.g. Ijspeert (2008)). These are "networks of neurons in spinal cords of vertebrates, capable of generating muscular activity in the absence of sensory feedback" (Holmes et al., 2006).

The above gait generation methods are all examples of continuous time methods. However, it is also possible to model the gait as a series of discrete events, which is done in the Switching Max-Plus-linear model (Lopes et al., 2009). This is a recently proposed method that models the gait as a series of touchdown and lift-off events, and is the gait generation method applied in this thesis.

Optimizing the gait parameters As mentioned earlier, a big challenge in the use of legged robotics is accurately choosing the parameters that describe the gait. Because of this, lots of research has been performed on how to determine the optimal parameters. This is done both for traditional optimization methods such as the Nelder-Mead method (e.g. Weingarten et al. (2004)), Powells method (e.g. Kim and Uther (2003)) and genetic algorithms (e.g. Chernova and Veloso (2004); Wolff et al. (2008)), as for learning methods, such as finite difference policy gradient Reinforcement Learning (RL) methods (e.g. Kohl and Stone (2004); Faber and Behnke (2007)) and Actor-Critic (AC) RL methods (e.g. Nakamura et al. (2007)). In this thesis the episodic Natural Actor-Critic (eNAC) method will be used, which is an RL method that can be seen as an evolution of the previously mentioned RL methods.

1-2 Outline of the thesis

The problem statement as considered in this thesis is:

Is it possible to learn the optimal gait parameters for a hexapod robot with respect to velocity and power consumption, where the gait is generated using the Switching Max-Plus-linear model, using the episodic Natural Actor-Critic method?

The structure of this thesis is shown in Figure 1-3. The theory behind the problem statement can be divided in two parts: the modelling of legged locomotion, which itself consists of constructing a dynamical model of a legged robot (Section 2-1) and the modelling of the gait generation (Section 2-2), and the learning method used to optimize the gait parameters (Chapter 3). Furthermore, in Chapter 4 the exact structure of the learning problem is defined and in Chapter 5 experiments are performed in which properties of the gait are optimized. Finally, in Chapter 6 the conclusions from this thesis are listed and possibilities on future work are mentioned.

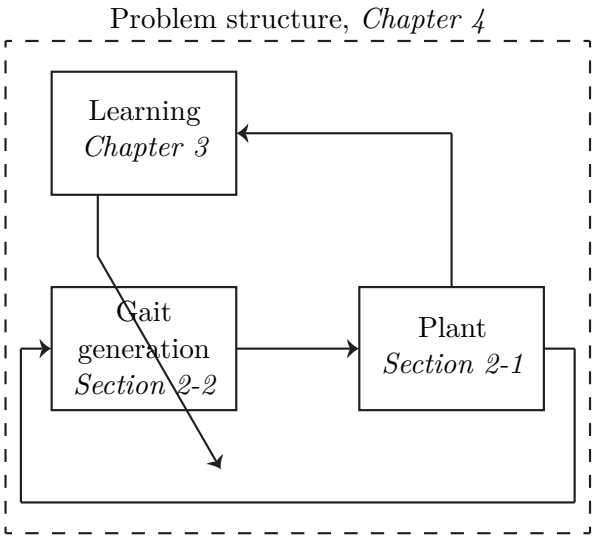


Figure 1-3: This figure shows how the thesis is structured. First, in Section 2-1 modelling of a legged robot is discussed, after which in Section 2-2 the gait generation used in this thesis is introduced. In Chapter 3 the learning method is explained and in Chapter 4 the theory of the previous chapters is combined to formulate the exact structure of the learning problem.

Chapter 2

Modelling of legged systems

Following the thesis structure as outlined in Section 1-2 this chapter deals with the problem of modelling legged systems. The problem of modelling can be divided into two sub-problems: how to model a legged robot and how to model gait generation. In Section 2-1 theory is presented with which it is possible to derive a simplified model of Zebro using the Euler-Lagrange equations. Furthermore, the inputs to the system and the external forces acting on the system are defined and integrated into the equations of motion. In Section 2-2 the Switching Max-Plus-linear model is introduced. This is a recently introduced method, based on the Max-Plus algebra. This method operates by enforcing the synchronization of multiple discrete event circuits, each composed of two events: leg lift-off and touchdown.

2-1 Modelling a hexapod robot

One way of modelling robots is by considering them as a collection of rigid bodies (Spong et al., 2006; Sicilliano et al., 2009), which is considered in Section 2-1-1. By applying the Euler-Lagrange equations to this collection the equations of motion can be derived in a systematic way. Furthermore, DC motor modelling is discussed and a contact model for the interaction between rigid bodies and non-movable objects, e.g. the ground, is introduced. In this thesis only the sagittal plane of Zebro is considered in order to simplify the problem. The model is therefore not intended to be a perfect model; it can be used for qualitative analysis, but not for quantitative analysis, i.e. the parameters found in optimization are not interchangeable between the model and Zebro. A second difference between the model and Zebro are the legs; in Zebro these are springy semi-circles made of PVC and in this thesis the legs are modelled as infinitely thin rigid bodies.

2-1-1 Rigid body dynamics

Before the Euler-Lagrange equations are introduced with which the equations of motion are derived, the necessary background is established: the configuration and workspace of a robot are defined and theory is presented with which it is possible to relate local coordinates to generalized coordinates. At the end of this section the equations of motion of the sagittal plane of Zebro are derived.

The configuration and workspace of a robot

The configuration space of a robot is a complete specification of the location of every point on the robot. The workspace is the total volume that can be reached by the end effector(s) of the robot (Spong et al., 2006). E.g. the configuration space of a robot that can move parallel to the axes of a Cartesian coordinate system is given by $\mathbb{R}^3$ and the workspace by $\mathbb{E}^3$.

This can be applied to Zebro (see Figure 1-1), which consists of a body and six recirculating legs, giving the configuration space: $SE(3) \times \mathbb{T}^6$. Here, $SE(n)$ denotes the Special Euclidean group[1] of order n. The workspace of Zebro is given by a subset of $\mathbb{E}^3$.

Local and generalized coordinate systems

Consider a system consisting of multiple rigid bodies, where each body has its own local coordinate frame. In order to relate the positions of all rigid bodies present in a system to each other, it is necessary to introduce a generalized coordinate frame. In Figure 2-1 two orthogonal coordinate frames are presented, where $\boldsymbol{O}_1$ is the local coordinate frame and $\boldsymbol{O}_0$ is the generalized coordinate frame.

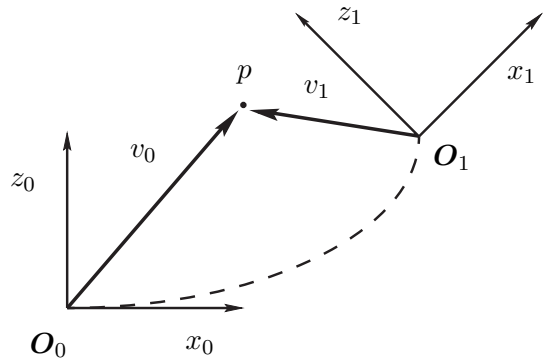


Figure 2-1: This figure shows a vector v_1 pointing to a point p in the local coordinate frame $\boldsymbol{O}_1$ and a vector v_0 pointing to the same point p but in the generalized coordinate frame $\boldsymbol{O}_0$. As can be seen the difference in origins is a pure translation and the difference in orientation of the frame is a rotation. The figure is adopted from Spong et al. (2006).

[1] A formal definition is given in Definition 1.

Local coordinates can be related to the generalized coordinates by a series of translations and rotations. Let p^0 be a point in coordinate frame $\boldsymbol{O}_0$ and p^1 a point in coordinate frame $\boldsymbol{O}_1$. The translation of a point in $\boldsymbol{O}_1$ to a point in $\boldsymbol{O}_0$ is given by

$$p^0 = d_1^0 \tag{2-1}$$

If coordinate frame $\boldsymbol{O}_1$ is rotated with respect to coordinate frame $\boldsymbol{O}_0$ a rotation matrix R_1^0 is introduced relating the frames to each other. Consider e.g. an orthogonal coordinate frame in $\mathbb{R}$ which has two possible translations, in x- and z-direction, and a rotation given by α. The corresponding rotation matrix is given by

$$R_1^0 = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{2-2}$$

where α is the angle of rotation from $\boldsymbol{O}_1$ to $\boldsymbol{O}_0$ and $R_1^0 \in SO(2)$, which denotes the Special Orthogonal group of order 2, or more general $R_1^0 \in SO(n)$ with order n.

By combining the translation and rotation of a coordinate frame, the concept of rigid motion can be defined (Spong et al., 2006).

Definition 1. Rigid motion. A rigid motion is an ordered pair (d, R), where $d \in \mathbb{R}^n$ and $R \in SO(n)$. The group of all rigid motions is known as the Special Euclidean Group and is denoted by $SE(n) : \mathbb{R}^n \times SO(n)$.

By combining translation and rotation the following equation is obtained

$$p^0 = R_1^0 p^1 + d_1^0 \tag{2-3}$$

It is possible to write this equation in matrix form using the homogeneous transformation matrix H_1^0

$$H_1^0 = \begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix}, \qquad R_1^0 \in SO(n), d_1^0 \in \mathbb{R}^n, \det(H_1^0) = 1 \tag{2-4}$$

resulting in

$$\begin{bmatrix} p^0 \\ 1 \end{bmatrix} = H_1^0 \begin{bmatrix} p^1 \\ 1 \end{bmatrix} \tag{2-5}$$

An important property of the transformation matrix is that multiple transformation matrices can be used in succession

$$H_2^0 = H_1^0 H_2^1 \tag{2-6}$$

Often it is necessary to relate the velocity of $\dot{p}^1$ to that of $\dot{p}^0$, which can be expressed by

$$\dot{p}^0(t) = \dot{R}_1^0(t)p^1(t) + R_1^0(t)\dot{p}^1(t) + \dot{d}_1^0(t) \tag{2-7}$$

The rotation matrix thus depends on time. If assumed that $\dot{p}^1$ is not moving with respect to its own coordinate frame, the equation reduces to[2]

$$\begin{aligned} \dot{p}^0 &= \dot{R}_1^0 p^1 + \dot{d}_1^0 \\ &= \dot{\alpha} \times R_1^0 p^1 + v \end{aligned} \tag{2-8}$$

where $\dot{\alpha}$ is the angular velocity of frame $\boldsymbol{O}_1$ with respect to frame $\boldsymbol{O}_0$, "$\times$" denotes the crossproduct and v denotes the linear velocity at which origin $\boldsymbol{O}_1$ is moving with respect to $\boldsymbol{O}_0$. This expression can be written in the form

$$\dot{p}^0 = \boldsymbol{J}(p^0)\dot{p}^1 \tag{2-9}$$

where $\boldsymbol{J}$ is known as the Jacobian.

Euler-Lagrange equations

To model the equations of motion of a collection of rigid bodies subjected to holonomic constraints[3] use is made of the Euler-Lagrange equations. Let the function $\mathcal{L}$ be the difference between the kinetic energy of a body and the potential energy of a body

$$\mathcal{L} = K(\boldsymbol{q}, \dot{\boldsymbol{q}}) - P(\boldsymbol{q}) \tag{2-10}$$

which is known as the Lagrangian of a system, with $\boldsymbol{q}$ the generalized coordinates.

The Euler-Lagrange equations are then defined as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = u_i + \tau_i, \qquad i = 1, \ldots, n \tag{2-11}$$

where n is the number of coordinates, u is the input to the system and τ are external forces and torques acting on the system.

When there are multiple bodies present in a system, the total kinetic energy of the system is given by

[2] For simplicity the time dependency is omitted.

[3] A holonomic constraint is of the form $0 = f(\boldsymbol{q})$, where $\boldsymbol{q}$ denotes the generalized coordinate vector.

$$K(\boldsymbol{p}, \dot{\boldsymbol{p}}) = \sum_{i=1}^{l} \frac{1}{2} \dot{p}_i^T M_i \dot{p}_i \tag{2-12}$$

where p_i denotes the local coordinates of body i, T the transpose of a vector, l the number of bodies in the system and M_i is the inertia matrix of body i, given by

$$M_i = \begin{bmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & J_i \end{bmatrix} \tag{2-13}$$

in case of $SE(2)$, where m is the mass of a body and J the moment of inertia.

The total potential energy of the system is given by

$$P(\boldsymbol{p}) = \sum_{i=1}^{l} P(p_i) \tag{2-14}$$

However, the kinetic and potential energy are given in local coordinates and need to be transformed to generalized coordinates in order to be used in the Euler-Lagrange equations as defined in Eq. (2-11). To transform the kinetic and potential energy to the generalized coordinates use is made of Eq. (2-4) and Eq. (2-9). The kinetic energy is now given by

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^T M(\boldsymbol{q}) \dot{\boldsymbol{q}} \tag{2-15}$$

where $\boldsymbol{q}$ denotes the generalized coordinates and $M(\boldsymbol{q})$ the new inertia matrix given by

$$M(\boldsymbol{q}) = \begin{bmatrix} M_1 & & \\ & \ddots & \\ & & M_l \end{bmatrix} \tag{2-16}$$

This matrix is symmetric and positive definite for each $\boldsymbol{q} \in \mathbb{R}^n$.

The total potential energy is given by $P(\boldsymbol{q})$ and the Lagrangian (see Eq. (2-10)) is given by

$$\mathcal{L} = K(\boldsymbol{q}, \dot{\boldsymbol{q}}) - P(\boldsymbol{q}) = \frac{1}{2} \dot{\boldsymbol{q}}^T M(\boldsymbol{q}) \dot{\boldsymbol{q}} - P(\boldsymbol{q}) \tag{2-17}$$

In order to derive the equations of motion of the total system a slightly different notation is introduced for the kinetic energy.

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \sum_{i,j}^{l} M_{i,j}(\boldsymbol{q}) \dot{q}_i \dot{q}_j \tag{2-18}$$

By implementing this expression into Eq. (2-17) and performing a series of algebraic manipulations it can be shown that the Lagrangian results in a system of the form (Spong et al., 2006)

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + G(\boldsymbol{q}) = \boldsymbol{u} + \boldsymbol{\tau} \tag{2-19}$$

where M is the inertia matrix, C is a matrix containing the Coriolis terms, G is a matrix related to the potential energy of the system, $\boldsymbol{u}$ represents the inputs to the system and $\boldsymbol{\tau}$ denotes the external forces and torques acting on the system.

The k, j-th element of the matrix $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is given by

$$\begin{aligned} C_{kj} &= \sum_{i=1}^{l} C_{ijk}(\boldsymbol{q})\dot{q}_i \\ &= \sum_{i=1}^{l} \frac{1}{2}\left\{ \frac{\partial M_{kj}}{\partial q_j} + \frac{\partial M_{ki}}{\partial q_j} - \frac{\partial M_{ij}}{\partial q_k} \right\}\dot{q}_i \end{aligned} \tag{2-20}$$

If the input $\boldsymbol{u}$ or the external forces and torques $\boldsymbol{\tau}$ are defined in local coordinates, it is necessary to transform them into generalized coordinates before implementing them in Eq. (2-19). This is done by pre multiplying them with the transpose of the Jacobian (see Eq. (2-9)), thus

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{q}) &= \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{u}(\boldsymbol{p}) \\ \boldsymbol{\tau}(\boldsymbol{q}) &= \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{\tau}(\boldsymbol{p}) \end{aligned} \tag{2-21}$$

Applying the Euler-Lagrange equations to Zebro

Zebro consists of a body and six legs, thus a total of seven rigid bodies. The six legs are related to the coordinate frame of the body and the coordinate frame of the body is related to the generalized coordinate frame. The local coordinate frames are located in the Centre of Mass (CoM) of each body, and are given by

$$p_i = [x_i, z_i, \theta_i]^T, \qquad i = 1, \ldots, 7 \tag{2-22}$$

The total kinetic energy of the system in local coordinates is given by

$$K(\boldsymbol{p}, \dot{\boldsymbol{p}}) = \sum_{i=1}^{7} \frac{1}{2}\dot{p}_i^T M_i \dot{p}_i \tag{2-23}$$

and the potential energy by

$$P(\boldsymbol{p}) = \sum_{i=1}^{7} P(p_i) = \sum_{i=1}^{7} m_i g z_i \tag{2-24}$$

where m_i is the mass of body i, g the acceleration due to gravity and z_i the height of the robot in local coordinates. However, the local coordinates must be translated to the generalized coordinates. In order to establish the generalized coordinates a schematic overview of the robot is given in Figure 2-2. In this figure only a single leg is drawn for simplicity.

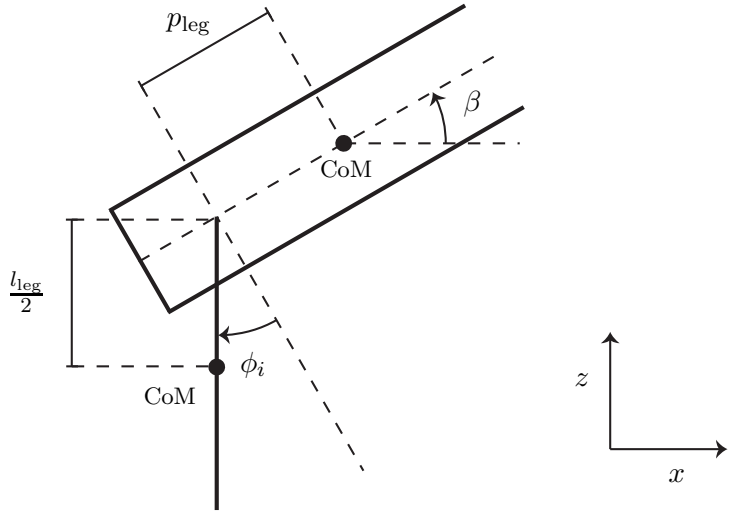


Figure 2-2: This figure shows a schematic overview of Zebro, in which the generalized coordinates $\boldsymbol{q}$ are depicted. For simplicity only a single leg is drawn. The variable p_{leg} denotes the distance between the point of rotation of the legs and the CoM of the body and l_{leg} denotes the leg length.

From Figure 2-2 it follows that the generalized coordinates are given by[4]

$$\boldsymbol{q} = [x, z, \beta, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6,]^T \tag{2-25}$$

The transformation matrix that relates the coordinate frame of the legs to the coordinate frame of the body is given by

$$\begin{aligned} H_l^b &= H^x H^\phi H^z \\ &= \begin{bmatrix} 1 & 0 & p_{\text{leg},i} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-\phi_i + \pi) & -\sin(-\phi_i + \pi) & 0 \\ \sin(-\phi_i + \pi) & \cos(-\phi_i + \pi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{l_{\text{leg}}}{2} \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -\cos(\phi_i) & -\sin(\phi_i) & p_{\text{leg},i} - \frac{l_{\text{leg}}}{2}\sin(\phi_i) \\ \sin(\phi_i) & -\cos(\phi_i) & -\frac{l_{\text{leg}}}{2}\cos(\phi_i) \\ 0 & 0 & 1 \end{bmatrix}, \qquad i = 1, \ldots, 6 \end{aligned} \tag{2-26}$$

[4] In subsequent chapters the generalized coordinates will be referred to as the internal state $\boldsymbol{q}$, and together with its derivatives $\dot{\boldsymbol{q}}$ as the internal state vector, in order to be in line with the remaining theory presented.

where H^z is the transformation in z_i-direction, H^ϕ is the rotation and H^x is the translation in x_i-direction. The variable p_{leg} denotes the distance between the point of rotation of the legs and the CoM of the body and l_{leg} denotes the leg length.

The transformation matrix that relates the coordinate frame of the body to the generalized coordinate frame is given by

$$H_b^0 = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & x \\ \sin(\beta) & \cos(\beta) & z \\ 0 & 0 & 1 \end{bmatrix} \tag{2-27}$$

where β is the rotation of the body.

Finally, the angles of rotation θ_i in generalized coordinates are given by

$$\theta_i = \begin{cases} \beta & \text{for } i = 1 \\ \beta + \phi_i & \text{for } i = 2, \ldots, 7 \end{cases} \tag{2-28}$$

The Jacobian of the system can then be found using Eq. (2-8) and the resulting system has the form as given in Eq. (2-19). The matrices as they appear in Eq. (2-19) and Eq. (2-21) are shown in Appendix A.

The inputs acting on the system are the torques applied by the DC motors driving the legs and are given in Section 2-1-2. The external forces acting on the system are the ground contact forces and are derived in Section 2-1-3.

2-1-2 DC motor modelling

In Zebro each leg is driven by a single DC motor, giving a total of six motors. The torques applied by these motors are the only inputs to the system. A simplified model of these electro motors is incorporated within the model, using the equations that are derived in this section. An overview of the situation in which a single DC motor is driving a single leg is given in Figure 2-3.

The dynamical equations for a DC motor are given by (Spong et al., 2006)

$$L\frac{dI}{dt} + RI = U - K_e\frac{d\phi}{dt} \tag{2-29}$$

where U is the applied voltage, L is the armature inductance, I is the armature current, R is the armature resistance, K_e is the speed constant and ϕ is the angle of rotation of the motor shaft.

If a single rotating leg is considered, as depicted in Figure 2-3, the equations of motion are given by

$$J\frac{d^2\phi}{dt^2} + b_m\frac{d\phi}{dt} + m_{\text{leg}}g\frac{l_{\text{leg}}}{2}\sin\phi = K_t I \tag{2-30}$$

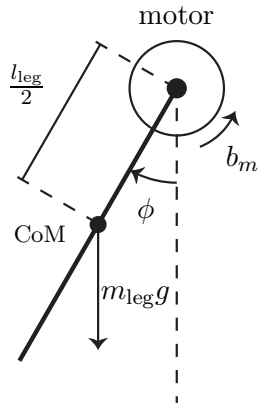


Figure 2-3: Driving a leg. In this figure an overview is given of a DC motor driving a leg.

with b_m the damping in the motor, m_{leg} the mass of the leg, g the acceleration due to gravity, l_{leg} the leg length, K_t the motor constant, and where

$$J = J_m + J_l + m_{\text{leg}}\left(\frac{l_{\text{leg}}}{2}\right)^2 \qquad [kgm^2] \tag{2-31}$$

is the total moment of inertia, J_m the moment of inertia of the motor and J_l the moment of inertia of the leg.

If the effect of the inductance is considered negligible, which is a commonly made assumption (e.g Franklin et al. (2006) and Grondman et al. (2011)), Eq. (2-29) and Eq. (2-30) can be combined into a single expression

$$J\frac{d^2\phi}{dt^2} + \left(b_m + \frac{K_t K_e}{R}\right)\frac{d\phi}{dt} + m_{\text{leg}}g\frac{l_{\text{leg}}}{2}\sin\phi = \frac{K_t}{R}U \tag{2-32}$$

If a gearbox is placed between the motor shaft and the leg, with the intention of reducing the speed of, and increasing the torque experienced by, the load, the equations have to be adapted

$$J\frac{d^2\phi}{dt^2} + \left(nb_m + n\frac{K_t K_e}{R} + b_g\right)\frac{d\phi}{dt} + m_{\text{leg}}g\frac{l_{\text{leg}}}{2}\sin\phi = n\frac{K_t}{R}U \tag{2-33}$$

where ϕ is now the angle of rotation of the leg and not of the motor shaft, b_g is the damping of the gearing and n is the reduction from motor shaft to load[5]. Furthermore, the total mass inertia J changes and is now given by

$$J = n^2 J_m + J_l + m_{\text{leg}}\left(\frac{l_{\text{leg}}}{2}\right)^2 + J_g \qquad [kgm^2] \tag{2-34}$$

where J_g is the mass inertia of the gearbox.

[5] A reduction of $n > 1$ leads to a load that is rotating slower than the motor shaft.

All the equations presented above hold for a single DC motor driving a single leg. For a system consisting of six DC motors, the equations are given by

$$J\frac{d^2\phi_i}{dt^2} + \left(nb_m + n\frac{K_tK_e}{R} + b_g\right)\frac{d\phi_i}{dt} + m_{\text{leg}}g\frac{l_{\text{leg}}}{2}\sin\phi_i = n\frac{K_t}{R}U_i, \qquad i = 1,\ldots,6 \quad \text{(2-35)}$$

These equations are already given in generalized coordinates, making it possible to easily integrate the motor equations into the equations of motion as derived in Section 2-1-1.

2-1-3 A ground contact model

A common way of modelling the leg-ground contact in legged robots is by introducing spring-damper systems at the tips of the legs (Silva et al., 2005; Holmes et al., 2006). By modelling the ground contact this way, the high impact forces that occur at touchdown are smoothed and large discontinuities in the model are avoided.

The ground contact forces can be divided into two parts: a force acting in z-direction, which accounts for the robot to be able to stand, and a force acting in x-direction, which accounts for the robot to be able to walk. It is therefore necessary to introduce two separate spring-damper systems. In Figure 2-4 the resulting reaction forces are drawn in a free-body diagram of a leg.

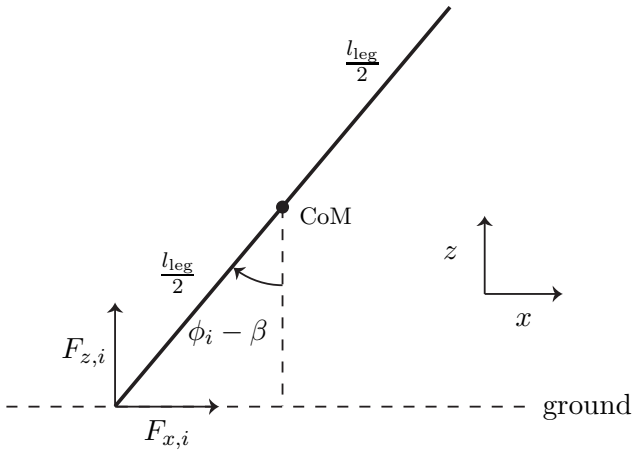


Figure 2-4: In this figure the reaction forces of the ground contact acting on a leg are shown. The forces are only present if the leg is touching the ground. The forces are given in a local coordinate frame and need to be translated to the generalized coordinate frame using the transpose Jacobian.

The forces are given by

$$\begin{aligned} F_{x,i} &= -K_{d,g,x}\dot{x}_{\text{tip},i} - K_{p,g,x}(x_{\text{tip},i} - x_{0,i}) \\ F_{z,i} &= -K_{d,g,z}\dot{z}_{\text{tip},i} - K_{p,g,z}(z_{\text{tip},i} - z_{\text{ground},i}), \qquad i = 1,\ldots,6 \end{aligned} \quad \text{(2-36)}$$

where $K_{d,g,x}$ is the damping in x-direction, $K_{p,g,x}$ is the stiffness, $\dot{x}_{\text{tip},i}$ is the velocity of the tip of leg i, $x_{\text{tip},i}$ is the position of the tip of leg i, $x_{0,i}$ is the x-position where the leg touched down and $z_{\text{ground},i}$ the height of the ground.

The position of the tips of the legs in generalized coordinates can be calculated by introducing a transformation matrix

$$H_t^l = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{l_{\text{leg}}}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{2-37}$$

that relates the tips of the legs to the CoM of the legs and subsequently use the transformation matrices as presented in Eq. (2-26) and Eq. (2-27). The velocities can be calculated using the Jacobian of the complete transformation matrix and to relate the forces to the generalized coordinates the transpose Jacobian of the complete transformation matrix must be used.

However, the forces are only present when a leg is in contact with the ground. If

$$z_{\text{tip},i} - z_{\text{ground},i} \leq 0, \qquad i = 1, \ldots, 6 \tag{2-38}$$

holds, leg i is touching the ground and the ground contact forces are active.

2-2 Gait generation using Max-Plus linear systems

In Section 1-1 it was stated that gait generation can roughly be divided into two parts: continuous time methods, such as Central Pattern Generators (CPGs) and the Spring-Loaded Inverted Pendulum (SLIP) method, and discrete event systems, such as the Switching Max-Plus-linear model. In this section the latter is discussed. As the name suggests, this gait generation method makes use of the Max-Plus algebra, where the state represents the time at which events occur, resulting in a purely event driven system.Max-plus algebras are introduced in Section 2-2-1 and switching Max-Plus linear systems, which are the main component of the gait generation, are discussed in Section 2-2-2.

2-2-1 Max-Plus algebra

The Max-Plus algebra was first named as such by Giffler (1960); Cunninghame-Green (1962), and is later extended by e.g. Cunninghame-Green (1979); Baccelli et al. (1992); Heidergott et al. (2006) and Butkovič (2010). In this section use is made of Baccelli et al. (1992) and Butkovič (2010), together with papers by Lopes et al. (2009, 2010), to establish the theoretical background of Max-Plus needed for the gait generation method.

The centre of Max-Plus algebra are two binary operations: $\oplus$ and $\otimes$. Let a and b be two scalars, then the operations $\oplus, \otimes$ are defined by

$$\begin{aligned} a \oplus b &= \max(a, b) \\ a \otimes b &= a + b \end{aligned} \tag{2-39}$$

Define $\varepsilon = -\infty$ and $e = 0$ as the neutral elements of the $\oplus$-operator and the $\otimes$-operator, respectively. An operation between a scalar a and the neutral element belonging to that operation, will result in scalar a, as is shown below.

$$\begin{aligned} a \oplus \varepsilon &= a \\ a \otimes e &= a \end{aligned} \tag{2-40}$$

Furthermore, let $\mathbb{R}_{\max} = \mathbb{R} \cup \varepsilon$. The set $\mathbb{R}_{\max}$ with the operations $\oplus$ and $\otimes$ is called the Max-Plus algebra, denoted by $\mathcal{R}_{\max} = (\mathbb{R}_{\max}, \oplus, \otimes, \varepsilon, e)$. It can be shown that $\mathcal{R}_{\max}$ is a commutative idempotent semiring, inheriting many tools from the linear algebra theory (Butkovič, 2010).

The theory of the scalar case can be extended to matrices and vectors. Let $A(i,j) = a_{ij}$ and $B(i,j) = b_{ij}$ be elements of matrices with compatible sizes. The max, plus and power operators for matrices are then defined by

$$\begin{aligned} (A \oplus B)_{ij} &= A(i,j) \oplus B(i,j) = a_{ij} \oplus b_{ij} := \max(a_{ij}, b_{ij}) \\ (A \otimes C)_{ij} &= A(i,j) \otimes C(i,j) = \bigoplus_{k=1}^{m} a_{ik} \otimes c_{kj} := \max_{k=1,\ldots,m} (a_{ik} + c_{kj}) \\ D^{\otimes k} &:= \underbrace{D \otimes D \otimes \ldots \otimes D}_{k-\text{times}} \end{aligned} \tag{2-41}$$

with $A, B \in \mathbb{R}_{\max}^{n \times m}, C \in \mathbb{R}_{\max}^{m \times p}$ and $D \in \mathbb{R}_{\max}^{n \times n}$.

The neutral elements for matrices and vectors are also known as the zero and identity matrices of Max-Plus, are build up from the neutral elements for the scalar case and are given by

$$\begin{aligned} \mathcal{E}(i,j) &= \varepsilon \\ E(i,j) &= \begin{cases} e & \text{if } i = j \\ \varepsilon & \text{if } i \neq j \end{cases} \end{aligned} \tag{2-42}$$

such that

$$\begin{aligned} A \oplus \mathcal{E} &= A \\ A \otimes E &= A \end{aligned} \tag{2-43}$$

holds.

A Max-Plus-linear system has the form

$$A \otimes x = b \tag{2-44}$$

where $A \in \mathbb{R}^{m \times n}$, referred to as the production or system matrix, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$.

Analogous to the system presented in Eq. (2-44), the following implicit system of equations can be defined

$$x = A \otimes x \oplus b \tag{2-45}$$

where $A \in \mathbb{R}_{\max}^{n \times n}$ and $b \in \mathbb{R}_{\max}^{n}$.

Furthermore, let

$$A^* = \bigoplus_{k=0}^{\infty} A^{\otimes k} \tag{2-46}$$

If A^* exists, the solution of the system presented in Eq. (2-45) is given by $x = A^* \otimes b$.

The matrix $D \in \mathbb{R}_{\max}^{n \times n}$ is called nilpotent if

$$\exists k < \infty, \forall p > k : D^{\otimes p} = \mathcal{E} \tag{2-47}$$

It always holds that if D is nilpotent $k < n$.

The eigenvectors v and eigenvalues λ of Max-Plus are defined in the traditional way

$$A \otimes v = \lambda \otimes v \tag{2-48}$$

where v denotes the steady-state behaviour of the system and λ the total cycle time.

Switching Max-Plus-linear systems are first proposed by van den Boom and De Schutter (2004, 2006) and are a form of Max-Plus-linear discrete event systems (MPL-DES) (see e.g. Heidergott et al. (2006)). In MPL-DES at every discrete event step k one or more events take place, think e.g. of a light being switched on or a valve that is closed. Such systems work in continuous time, but are driven by discrete events.

A set of implicit Max-Plus-linear systems is introduced, given by

$$x(k) = G \otimes x(k) \oplus H \otimes x(k-1) \tag{2-49}$$

where k denotes the discrete event step, $x \in \mathbb{R}^n$ is the state vector. The matrices $G, H \in \mathbb{R}^{n \times n}$ are the system matrices.

It is shown in Section 2-2-2, that this system can be written into an explicit set of switching Max-Plus-linear systems, under the correct conditions of G and H, given by

$$x(k) = A \otimes x(k-1) \tag{2-50}$$

where $A \in \mathbb{R}^{n \times n}$ is the system matrix, defining the order of events.

2-2-2 Switching Max-Plus-linear model

The gait generated method discussed in this section, the Switching Max-Plus-linear model, is based on the Max-Plus algebra. To be more precise, the Switching Max-Plus-linear model is a switching MPL-DES (Section 2-2-1). In Max-Plus-linear systems it is possible for the MPL-DES to change between different modes of operation, by changing the system matrix A. As mentioned in Section 2-2, the first application of the Max-Plus algebra in legged locomotion is done by Lopes et al. (2009, 2010), and the majority of the theory presented in this section is adopted from those two sources. The different modes of operation present in a switching Max-Plus-linear system are in case of the Switching Max-Plus-linear model the different gaits with which a legged robot can walk, e.g. a trot or a gallop.

The great advantage of modelling the gait generation as a MPL-DES is that the generated gait is completely driven by the lift-off and touchdown events of the legs. Because the gait is completely event driven, it is relatively easy to ensure stability[6] of the robot by synchronizing (pairs of) legs with each other. For instance, when considering a biped robot, the stance leg should only lift-off from the ground if the swing leg has touched the ground, to guarantee the robot does not fall over[7]. As a result of requiring the synchronization of legs, the robot will never enter an aerial phase in which all legs are from the ground.

In order to apply the Max-Plus algebra to legged locomotion, it is necessary to define the lift-off and touchdown events in a formal way, which is done in subsection "Max-Plus gait scheduler". However, first the control structure of the Switching Max-Plus-linear model is established in Figure 2-5 in order to derive the complete gait generation in a structured way. The structure and equations are derived for robots with recirculating legs, but can be adapted for robots with other types of legs. However, the equations as given for the continuous time scheduler might require an extra map to convert the phase into appropriate configuration space coordinates.

Supervisory control

In Figure 2-5, the first block, the supervisory control block, makes decisions regarding which type of gait to use. As shown in Section 1-1 the gait of animals is optimized with respect to the energetic cost of the movement, at a certain speed. A similar approach can be used within the supervisory block to switch gaits. In Lopes et al. (2010) it was shown numerically that switching between gaits can be done without destabilizing the robot.

[6] In this section only static stability is considered, not dynamic stability.

[7] It is assumed that the robot is always stable when both legs are touching the ground.

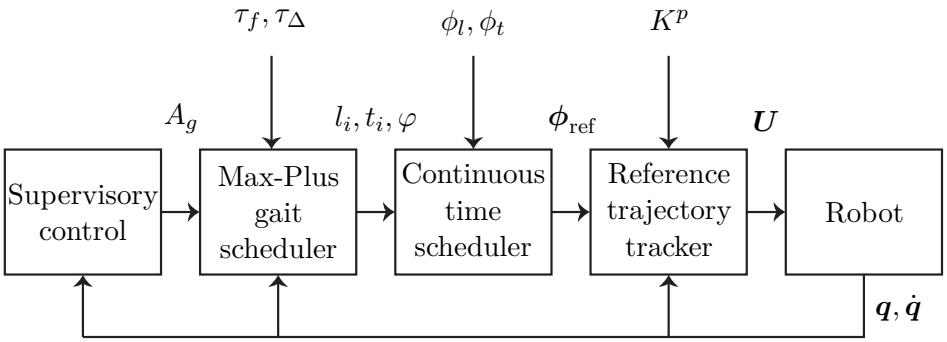


Figure 2-5: The control structure for a legged robot with recirculating legs, using the Switching Max-Plus-linear model. The control structure consists of five blocks. First of all there is the supervisory control block, which determines the gait to be used and outputs the gait matrix. Second, there is the Max-Plus gait scheduler, which determines when events take place. Furthermore, it outputs the phase of the Max-Plus gait generation. Third, there is the continuous time scheduler, which transforms the discrete events to continuous time. Fourth, there is a reference trajectory tracker, which task it is to make sure the legs follow the reference trajectory. The signal going from the reference trajectory tracker to the robot is a control signal for the DC motors driving the legs, and is denoted by $\boldsymbol{U}$. The last block present is the robot itself, outputting the internal states $\boldsymbol{q}, \dot{\boldsymbol{q}}$. Furthermore, there are feedback loops present from the robot to the reference trajectory tracker, to the Max-Plus gait scheduler and to the supervisory control block. The figure is adopted from Lopes et al. (2009).

If two or more legs are present in a system it is possible to define sets of legs that move at the same time. Thus, it is possible to define different types of gaits by defining different leg sets. For leg sets containing only a single leg, let

$$\{L_1\} \prec \ldots \prec \{L_j\} \tag{2-51}$$

with $j = 1, \ldots, m$ and m the number of leg sets. The symbol "$\prec$" denotes that leg set j is only allowed to lift-off after leg set $j-1$ touched down. This notation can be extended to leg sets containing multiple legs by

$$\{L_{1,1}, \ldots, L_{1,a}\} \prec \ldots \prec \{L_{j,1}, \ldots, L_{j,b}\} \tag{2-52}$$

with $j = 1, \ldots, m$ and m the number of leg sets. The symbols a and b denote the number of legs within a leg set. For instance, a tripod gait for a hexapod robot is defined by $\{1, 4, 5\} \prec \{2, 3, 6\}$, where the numbers denote the leg numbers. For this robot the leg numbering starts at the front left leg and continues from left to right, front to back.

Max-Plus gait scheduler

The second block is the Max-Plus gait scheduler, which outputs the discrete event times for lift-off and touchdown. Let $l_i(k)$ be the time instant leg i lifts off from the ground

and let $t_i(k)$ be the time instant leg i touches the ground, both for the k^{th} iteration. If the robot is moving with an alternating swing/stance gait, it can be required that the time instant a leg touches the ground must equal the time instant it lifted off from the ground for the last time, plus the time the leg is in the air. This relation is given by

$$t_i(k) = l_i(k) + \tau_f \tag{2-53}$$

where τ_f is the flight time of the leg, i.e. the time the leg is in the air.

Analogously it is possible to define the time instant a leg lifts off from the ground by

$$l_i(k) = t_i(k-1) + \tau_g \tag{2-54}$$

where τ_g is the ground time of the leg, i.e. the time the leg is on the ground.

Synchronization of legs can be achieved by introducing an extra parameter τ_Δ, which is known as the double stance time. The double stance time is defined as the time between the touchdown of leg i and the lift-off of leg j, i.e. it is not possible for leg j to lift-off before the double stance time is passed. The double stance time can be expressed in terms of the flight time τ_f, the ground time τ_g and the number of leg sets m, and is given by

$$\tau_\Delta = \frac{\tau_g + \tau_f}{m} - \tau_f \tag{2-55}$$

Analogously it is possible to relate the ground time τ_g to the flight time τ_f, the double stance time τ_Δ and the number of leg sets by

$$\tau_g = m(\tau_f + \tau_\Delta) - \tau_f \tag{2-56}$$

There are thus only three parameters of importance: the gait, as it gives the number of leg sets, the flight time and the double stance time.

The lift-off time of leg i can be written in terms of the touchdown time of leg i and the touchdown time of leg j using Eq. (2-54).

$$\begin{aligned} l_i(k) &= \max(t_i(k-1) + \tau_g, t_j(k-1) + \tau_\Delta) \\ &= \begin{bmatrix} \tau_g & \tau_\Delta \end{bmatrix} \otimes \begin{bmatrix} t_i(k-1) \\ t_j(k-1) \end{bmatrix} \end{aligned} \tag{2-57}$$

This equation must be read as follows: leg i can only lift-off if it has been on the ground for at least τ_g seconds and if leg j is on the ground for at least τ_Δ seconds, depending on which of them happens latest in time. If both conditions are satisfied leg i will lift-off

from the ground. This property is useful when, for instance, leg j is stopped while in the air; it is then not possible for leg i to lift-off. If the robot was stable it will remain stable.

Analogously to Eq. (2-57) it is possible to define the touchdown time $t_i(k)$ in terms of the lift-off times, using Eq. (2-53), which results in

$$\begin{aligned} t_i(k) &= \max(l_i(k) + \tau_f, l_j(k) + \tau_\Delta) \\ &= \begin{bmatrix} \tau_f & \tau_\Delta \end{bmatrix} \otimes \begin{bmatrix} l_i(k) \\ l_j(k) \end{bmatrix} \end{aligned} \tag{2-58}$$

If Eq. (2-57) and Eq. (2-58) are extended to an n-legged system, the following discrete event state vector is obtained

$$x(k) = [\underbrace{t_1(k), \ldots, t_n(k)}_{t(k)} \underbrace{l_1(k), \ldots, l_n(k)}_{l(k)}]^T \tag{2-59}$$

with $x(k) \in \mathbb{R}_{\max}^{2n}$ and where T denotes the transpose of a vector.

Eq. (2-53) and Eq. (2-54) can now be rewritten as a system of $2n$ dimensions, resulting in

$$\begin{bmatrix} t(k) \\ l(k) \end{bmatrix} = \begin{bmatrix} \mathcal{E} & \tau_f \otimes E \\ \mathcal{E} & \mathcal{E} \end{bmatrix} \otimes \begin{bmatrix} t(k) \\ l(k) \end{bmatrix} \oplus \begin{bmatrix} E & \mathcal{E} \\ \tau_g \otimes E & E \end{bmatrix} \otimes \begin{bmatrix} t(k-1) \\ l(k-1) \end{bmatrix} \tag{2-60}$$

The identity matrices E have been added to the set of equations to implement the constraints $t(k) \geq t(k-1)$ and $l(k) \geq l(k-1)$. From this system of equations it follows furthermore that all legs rotate with the same period of at least $\tau_g + \tau_f$.

It is assumed that leg synchronization can be achieved by enforcing a relation between the current lift-off time of a leg and the previous touchdown times of the other legs. This is done by introducing the additional matrices P and Q, which are defined later, resulting in the synchronized system:

$$\begin{bmatrix} t(k) \\ l(k) \end{bmatrix} = \begin{bmatrix} \mathcal{E} & \tau_f \otimes E \\ P & \mathcal{E} \end{bmatrix} \otimes \begin{bmatrix} t(k) \\ l(k) \end{bmatrix} \oplus \begin{bmatrix} E & \mathcal{E} \\ \tau_g \otimes E \oplus Q & E \end{bmatrix} \otimes \begin{bmatrix} t(k-1) \\ l(k-1) \end{bmatrix} \tag{2-61}$$

which can be written as

$$x(k) = A_0 \otimes x(k) \oplus A_1 \otimes x(k-1) \tag{2-62}$$

and has the form of an implicit switching Max-Plus linear system (Eq. (2-49)).

This system can be written as an explicit switching Max-Plus linear system, using the knowledge that a sufficient condition for A_0^* to exist is that P is nilpotent (proof given in Lopes et al. (2010)), and the equations are given by

$$\begin{aligned} x(k) &= A_0^* \otimes A_1 \otimes x(k-1) \\ &= A_g \otimes x(k-1) \end{aligned} \tag{2-63}$$

where the state $x(k)$ contains the time instants at which events occur for the k^{th} time and $A_g = A_0^* \otimes A_1 \in \mathbb{R}^{2n \times 2n}$ is the gait matrix, with n the number of legs. This matrix is referred to as the Max-Plus gait scheduler as it schedules when events take place. The size of the gait matrix A_g is constant, assuming no change in the number of legs takes place, but the way it is filled up with values depends on the type of gait used.

The matrices P and Q are used to encode the gaits, where the gait is defined in terms of the leg sets (see subsection "Supervisory control"). Consider the leg set sequence

$$\{L_1\} \prec \ldots \prec \{L_j\} \tag{2-64}$$

with $j = 1, \ldots, m$ and m the number of leg sets. Furthermore, assume that the double stance time τ_Δ is required between the moment of touchdown of leg set $j-1$ and the moment of lift-off of leg set j, then the matrices can be constructed as follows. The start matrices are $P = \mathcal{E}$ and $Q = \mathcal{E}$ and for each pair $\{L_i\} \prec \{L_{i+1}\}$, with $i = 1, \ldots, n-1$ and n the number of pairs, an entry is added to the matrix P in row L_{i+1} and column L_i, given by

$$[P]_{L_{i+1},L_i} = \tau_\Delta \tag{2-65}$$

To enforce the synchronization of the full cycle, the first and last pair of legs L_1 and L_m are added to matrix Q such that

$$[Q]_{L_1,L_m} = \tau_\Delta \tag{2-66}$$

When there are multiple legs present within a single leg set, entries must be added to any combination of legs, in both P and Q, following

$$\begin{array}{cc} [P]_{L_{2,1},L_{1,1}} = \tau_\Delta & [Q]_{L_{1,1},L_{m,1}} = \tau_\Delta \\ [P]_{L_{2,1},L_{1,2}} = \tau_\Delta & [Q]_{L_{1,1},L_{m,2}} = \tau_\Delta \\ \vdots & \vdots \end{array} \tag{2-67}$$

For a hexapod robot walking with a tripod gait the leg sets are given by $\{1,4,5\} \prec \{2,3,6\}$ (see subsection "Supervisory control") and the P and Q matrices are given by (Lopes et al., 2010)

$$P = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \tau_\Delta & \varepsilon & \varepsilon & \tau_\Delta & \tau_\Delta & \varepsilon \\ \tau_\Delta & \varepsilon & \varepsilon & \tau_\Delta & \tau_\Delta & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \tau_\Delta & \varepsilon & \varepsilon & \tau_\Delta & \tau_\Delta & \varepsilon \end{bmatrix} \qquad Q = \begin{bmatrix} \varepsilon & \tau_\Delta & \tau_\Delta & \varepsilon & \varepsilon & \tau_\Delta \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \tau_\Delta & \tau_\Delta & \varepsilon & \varepsilon & \tau_\Delta \\ \varepsilon & \tau_\Delta & \tau_\Delta & \varepsilon & \varepsilon & \tau_\Delta \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix} \tag{2-68}$$

If the results above are used in combination with the conclusion drawn under Eq. (2-46), it is possible to define the eigenvector and eigenvalue of the gait matrix A_g. Let $\mathbf{0}$ be a zero vector with size $2n \times 1$, the eigenvector of gait matrix A_g can be found using

$$v = A_g \otimes \mathbf{0} \tag{2-69}$$

The corresponding eigenvalue can be found using

$$\lambda = \max(A_g \otimes v - v) \tag{2-70}$$

where λ is the eigenvalue.

In Baccelli et al. (1992) it is shown that the eigenvalue of the system matrix can be interpreted as the cycle time of the underlying system. Applied to the Switching Max-Plus-linear model this means that the time it takes for a leg to complete a full circular movement equals the eigenvalue of A_g.

The phase of the legs in time, denoted by φ and referred to as *the phase of the Max-Plus gait generation*, can be related to the cycle time. However, $\varphi \in \mathcal{S}^1$ and $t, \lambda \in \mathbb{R}$, thus

$$\varphi = f_{\mathcal{S}^1}\left(\frac{t}{\lambda}\right) \tag{2-71}$$

where the function $f_{\mathcal{S}^1}$ projects the real number into the circle. In practice, this function can be written as the modulo operator.

Continuous time scheduler

The gait generation method as presented until now is completely event driven; it does not specify the trajectories the legs of the robot should follow, it only tells when a certain even must take place. Therefore, it is necessary to introduce a continuous time scheduler, which transforms the discrete events to a continuous time representation by defining reference trajectories for the legs.

The reference trajectory function for an n-legged robot is defined by $\boldsymbol{\phi}_{\text{ref}} : \mathbb{R}^+ \times \mathbb{R}^{2n}_{\max} \to (\mathcal{S}^1)^n$ and the reference trajectory is given by

$$\phi_{\text{ref},i}(\varphi) = \begin{cases} \dfrac{\phi_l(t_i(k_{2i-1}) - \varphi) + (\phi_t + 2\pi)(\varphi - l_i(k_{2i}))}{t_i(k_{2i-1}) - l_i(k_{2i})} \\ \qquad\qquad \text{if } \varphi \in [l_i(k_{2i}), t_i(k_{2i-1})) \\ \dfrac{\phi_t(l_i(k_{2i}+1) - \varphi) + \phi_l(\varphi - t_i(k_{2i-1}))}{l_i(k_{2i}+1) - t_i(k_{2i-1})} \\ \qquad\qquad \text{if } \varphi \in [t_i(k_{2i-1}), l_i(k_{2i}+1)) \end{cases} \tag{2-72}$$

where ϕ_l and ϕ_t are the angles of rotation of the legs at the lift-off and touchdown event[8], respectively, and the input φ is the phase of the Max-Plus gait generation. Furthermore, the equation $\phi_t < \phi_l$ must be satisfied at all times.

Reference trajectory tracker

The reference trajectory tracker ensures the legs follows their reference trajectories and can be any type of controller, e.g. a PD-controller. It is here assumed that the controller can be parametrized by a parameter set K^p.

Robot

The final block illustrated in Figure 2-5, represents the robot (or in more general terms: the plant), which in this thesis is Zebro. In this section a robot with recirculating legs is assumed, but the method can also be applied to other legged robots.

Feedback and parameter adaptation

The feedback loop present in Figure 2-5 serves three purposes. First, it loops back the actual positions of the legs to the reference trajectory tracker in order to control the positions of the legs. Second, there is a feedback loop to the Max-Plus gait scheduler. This loop is used to update the calculated future lift-off and touchdown event times using information about the actual times of previous events. This is necessary for cases in which, for instance, one of the legs is stopped while in the air, where in order to guarantee stability of the robot the stance legs should not lift-off. The final feedback loop goes back to the supervisory control in which the decision to change gaits is made based on e.g. energetic costs of the gait.

The complete gait generation is parametrized by only a few parameters as follows from the previous sections, which are: the gait matrix A_g, the flight time τ_f and double stance time τ_Δ, the lift-off angle ϕ_l and touchdown angle ϕ_t and finally the controller parameters K^p. The Switching Max-Plus-linear model offers several useful properties regarding parameter adaptation. First, the gait matrix A_g can be changed safely due to the inherited properties of the Max-Plus algebra (Lopes et al., 2010). This includes changing the parameters within the matrix, τ_f and τ_Δ, as well as the structure of the matrix itself. However, the most important realization is that the lift-off and touchdown angles can safely be changed to generate new reference trajectories. Adapting the reference trajectories is necessary, for instance, when one wants to steer the robot, which can be done by introducing different angle offsets for the legs on either side of the robot. More importantly, the lift-off and touchdown angles have to be adapted according to the state of the robot. The relation between states and touchdown and lift-off angles will be examined in Chapter 5.

[8] The lift-off and touchdown angles are the desired angles, not the measured angles.

Overall, it can be said that the gait matrix A_g is changed in case of a gait switch, the controller parameters K^p can be fixed, the parameters ϕ_l and ϕ_t need to be adapted based on the state of the robot and the parameters τ_f and τ_Δ need to be adapted in case of external disturbances (such as a leg that is stopped)[9].

2-3 Conclusions

In this chapter the Euler-Lagrange method was used to model Zebro as a system of seven rigid bodies. Although it is necessary to make some crude assumptions regarding the geometry of the robot, it is a widely used and accepted method for robot modelling. Furthermore, DC motor modelling and a method of modelling the ground contact of a rigid body were discussed. The latter relies on the approximation of the ground contact by two sets of linear springs and dampers, which offers a trivial way of implementing contact forces into the model of Zebro.

In the second part of the chapter a gait generation method was discussed based on the Max-Plus algebra, the so-called Switching Max-Plus-linear model. The core of this gait generation are two events: the lift-off and touchdown of the legs. The timings at which these events must take place are generated based on the type of gait used, the time the legs are in the air and the time that all legs are simultaneously on the ground. By assuming recirculating legs, such as the ones present on Zebro, it is possible to generate the reference trajectories for the legs based on the desired angles of the legs at the lift-off and touchdown events. Because of the intuitive way of defining the gait, combined with only five parameters that describe the complete gait, the Switching Max-Plus-linear model is a method with large potential in the domain of legged robotics with a morphology similar to that of Zebro. Furthermore, changing the gait can easily be done without destabilizing the robot. The question on how to find the optimal parameters is further investigated in Chapter 3, where a learning method is introduced, and Chapter 4, in which the structure of the problem is defined.

[9] In this thesis no gait switching or external disturbances are assumed.

Chapter 3

Reinforcement Learning

As follows from the outline of the thesis in Section 1-2 this chapter deals with a learning method that is capable of optimizing the gait of a legged robot. To be more precise, the episodic Natural Actor-Critic (eNAC) method is used to learn an optimal gait. This is a model-free Reinforcement Learning (RL) technique, which has successfully been applied to high dimensional systems (Peters and Schaal, 2008b) as, in general, encountered when dealing with legged robots. RL is a learning framework inspired by the way animals learn to deal with new situations. In RL the learning agent is not told what to do in a specific situation; it has to discover this on its own via trial-and-error. It therefore receives a numerical reward for every state transition, the better the transition the higher the reward, and tries to maximize the return over the long run. The trial-and-error nature together with the maximization of the return over the long run are important properties of RL. In Section 3-1 an introduction to RL containing the important aspects necessary for this thesis is given, after which in Section 3-2 the eNAC method is discussed in detail.

3-1 RL framework

In this section a short overview of the RL framework is given. This overview will only contain the information necessary for this thesis and is therefore not exhaustive. The majority of the information given is adapted from Sutton and Barto (1998); Szepesvári (2010) and Sigaud and Buffet (2010). It must be noted that the theory in this section is only given for the discrete time case.

In RL, of which the general structure is given in Figure 3-1, the *agent* receives at each time step k a representation of the *state* s_k of the *environment* and performs an *action* a_k thereby changing the state of the environment to a new state s_{k+1}. The environment responds by giving the agent a *reward* r_{k+1}, according to a reward function, which is based on how good that particular action was in that particular state. The selection of

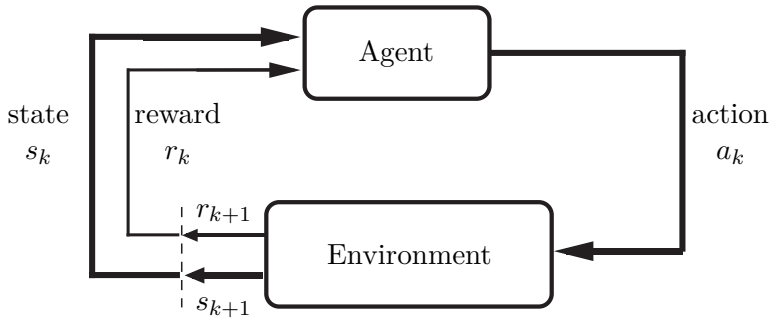


Figure 3-1: The general Reinforcement Learning structure. The agent outputs an action a_k to the environment. The environment responds by presenting the agent a new state s_{k+1} and a reward r_{k+1}. The dotted line denotes the difference between time step k and time step $k+1$. The figure is adopted from Sutton and Barto (1998).

an action is done according to a *policy*. The terms denoted in italic will be explained more thoroughly in the next paragraphs.

State The state $s_k \in \mathbb{S}$, where $\mathbb{S}$ is the set of possible states, is defined by a signal from the environment to the agent, representing (some) properties of the environment.

Action The action $a_k \in \mathbb{A}(s_k)$, where $\mathbb{A}(s_k)$ is the set of actions available in state s_k, is the output of the agent. The action taken influences the state of the environment and the selection of an action is done according to a policy.

Policy A stochastic policy, denoted by $\pi(a_k = a|s_k = s)$, gives the probability that $a_k = a$ if $s_k = s$. A special case of the stochastic policy is the case in which the probability that $a_k = a$ if $s_k = s$ equals 1, resulting in a deterministic policy. An optimal policy, denoted by π^*, is the policy that corresponds to the greatest received return by the agent. It must be noted that a policy does not have to be unique, i.e. multiple policies can result in the same return.

Environment The environment is defined as everything outside of the agent. It receives an action from the agent, and outputs a new state and reward to the agent. In the deterministic setting the next state is a function of the current state and action, denoted by $s_{k+1} = X(s_k, a_k)$. In case of a stochastic environment the probability of the next state is given by $P(s_{k+1} = s'|s_k, a_k)$, where s' denotes the state at the next time step.

If the environment is deterministic, taking the same action in the same state at two different steps results in the same next states and rewards. In the non-deterministic case, taking the same action in the same state at two different steps may result in different next states and/or different rewards.

Agent The agent is the learner and decision maker in the RL framework. It can influence the state of the environment by performing an action and receives a reward based on the state transition. The goal of the agent is to maximize the return, which is a function of the rewards, over a trajectory generated by a policy. Maximization of the return is done by adapting the policy the agent follows until an optimal policy is found, at which point the return does not increase anymore.

Reward The agent receives a reward from the environment, based on how good or how bad a particular action in a particular state was. At each time step k the reward is a number r_k, according to the reward function $\rho : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$.

Return The agent's goal is to maximize the return J_k, where the return is defined as a function of the reward sequence starting at state s_k. In the simplest case the return is the sum of all the rewards received:

$$J_k(\pi) = E_\pi \left\{ \sum_{n=0}^{N-1} r_{n+k+1} \Big| s_k = s \right\} \tag{3-1}$$

where $E_\pi\{\cdot\}$ denotes the expected value when following policy π, k denotes the current time step and N is the amount of steps taken after the current time step[1].

A sequence of actions terminating at a finite time step $N < \infty$ is called an episodic task. The state corresponding to this final time is called the terminal state. It is not possible to leave this state, regardless of the action performed; the state must be reset to a (new) starting state before the next episode is started. The value of the final state is always zero. However, this way of calculating the return is only feasible in applications where a final time step N can be defined.

In many tasks it is not possible to define a terminal state; these tasks are called continuous tasks. If the return is then calculated as in Eq. (3-1), with $N = \infty$, the sum can grow to infinity (e.g. if the agent receives a reward of 1 at every time step). The solution to this problem is to define the return in a different way, making sure the return is bounded. There are several possibilities to do this, of which two will be discussed here: the discounted return and the average return. The discounted return is calculated with

$$J_k(\pi) = E_\pi \left\{ \sum_{n=0}^{N-1} \gamma^n r_{n+k+1} \Big| s_k = s \right\} \tag{3-2}$$

where γ is the discount factor, $\gamma \in [0, 1)$[2].

The discount factor determines the current value of rewards received in the future. The smaller the discount factor, the less the current value of a reward received in the future.

[1] This return is defined for the stochastic case. In the deterministic case the return is always the same when in a given state performing a given action and $E_\pi\{\cdot\}$ can therefore be dropped.

[2] If the discount factor would equal 1, the undiscounted return, as defined in Eq. (3-1), is obtained. Therefore, the value of the discount factor is limited in the range $[0, 1)$.

Theorem 1. *For both episodic tasks and continuous tasks, with $N = \infty$, the discounted return has a finite value, as long as $\gamma < 1$ and $\rho(s, a)$ is bounded.*

Proof. Assume $|\rho(s, a)| \leq M, \forall s, a$. Let the worst case be given by

$$\begin{aligned} J_k(\pi) &= E_\pi \left\{ \sum_{n=0}^{N-1} \gamma^n r_{n+k+1} \Big| s_k = s \right\} \\ &\leq E_\pi \left\{ \sum_{n=0}^{N-1} \gamma^n M \Big| s_k = s \right\} \\ \gamma J_k(\pi) &\leq E_\pi \left\{ \sum_{n=0}^{N-1} \gamma^{n+1} M \Big| s_k = s \right\} \end{aligned} \tag{3-3}$$

then

$$J_k(\pi) - \gamma J_k(\pi) = J_k(\pi)(1 - \gamma) = \left(1 - \gamma^N\right) E_\pi \{M | s_k = s\} \tag{3-4}$$

thus $J_k(\pi) = E_\pi \{M | s_k = s\} \frac{1-\gamma^N}{1-\gamma}$. If $\gamma \in [0, 1)$

$$\lim_{n \to \infty \frac{1-\gamma^N}{1-\gamma}} E_\pi \{M | s_k = s\} = \frac{1}{1-\gamma} E_\pi \{M | s_k = s\} \tag{3-5}$$

□

Another way of preventing the return of going to infinity is by using the average return, which is defined by

$$J_k(\pi) = \lim_{N \to \infty} \frac{1}{N} E_\pi \left\{ \sum_{n=0}^{N-1} r_{n+k+1} \Big| s_k = s \right\} \tag{3-6}$$

where N denotes the final time step.

Theorem 2. *The convergence of the average return to a finite value follows directly from the law of large numbers and the assumption that $\rho(s, a)$ is bounded.*

Proof. Let $\rho_1(s, a) + \ldots + \rho_n(s, a)$ be a sequence of independent random rewards with mean $\langle \rho_i(s, a) \rangle = \overline{\rho(s, a)}$ and variance σ^2. Define $J_k(\pi) = (\rho_1(s, a) + \ldots + \rho_n(s, a))/n$. If $n \to \infty$, the sample mean $\langle J_k(\pi) \rangle$ equals the population mean $\overline{\rho(s, a)}$ of each reward:

$$\begin{aligned}\langle J_k(\pi)\rangle &= \left\langle \frac{\rho_1(s,a)+\ldots+\rho_n(s,a)}{n} \right\rangle \\ &= \frac{1}{n}(\langle\rho_1(s,a)\rangle+\ldots+\langle\rho_n(s,a)\rangle) \\ &= \frac{n\overline{\rho(s,a)}}{n} \\ &= \overline{\rho(s,a)}\end{aligned} \tag{3-7}$$

□

The Markov property In RL, the policy and reward function are assumed to be functions of s_k, not $s_{k-1}, s_{k-2}, \ldots$. In order to take the best actions and receive the greatest return, as much (useful) information as possible about the environment should be included in the current state. This is the case if the state signal has the Markov property, as follows from the definition given by Sutton and Barto (1998):

Definition 2. The Markov Property. When the environment is described by a state signal that summarizes all past sensations compactly, yet in such a way that all relevant information is preserved, the state signal is said to have the Markov property.

In the most general case, the way an environment responds at time $k+1$ to the action taken at time k, depends on everything that has happened before. The dynamics of the environment can only be defined by the complete probability function

$$P\{s_{k+1}=s', r_{k+1}=r|s_k, a_k, r_k, \ldots, s_0, a_0\} \tag{3-8}$$

for the next state s', reward r, and all possible values of the past events: $s_k, a_k, r_k, \ldots, s_0, a_0$. In the specific case in which the state signal has the Markov property, the environment's response at $k+1$ depends only on the state s_k and action a_k, and the environment's dynamics can be defined by

$$P\{s_{k+1}=s', r_{k+1}=r|s_k, a_k\} \tag{3-9}$$

Markov Decision Process An RL task that satisfies the Markov property can be modelled as a Markov Decision Process (MDP). If the state and action spaces are finite, the task is called a finite MDP. Given a state s_k and action a_k, the state transition probability is given by

$$\mathcal{P}^a_{ss'} = P\{s_{k+1}=s'|s_k=s, a_k=a\} \tag{3-10}$$

The expected value of the next reward is given by

$$\mathcal{R}_s^a = E\{r_{k+1}|s_k = s, a_k = a, s_{k+1} = s'\} \tag{3-11}$$

In practice it is never possible for the agent to observe the complete state of the environment, due to e.g. sensor limitations and noise. In these cases, the environment may still have the Markov property, but the agent only observes parts of the environment, making the process a so-called Partially Observable Markov Decision Process (POMDP)[3].

Value functions Most RL algorithms are based on estimating value functions, which are functions of the state, or state-action pair, that estimate how good it is to be in a given state, or state-action pair, following policy π. The term "how good" is defined in terms of the expected return. There are two types of functions: the state value function and the state-action value function.

The state value function, denoted by $V^\pi(s)$, is the expected return when starting in s_k and following policy π thereafter.

$$V^\pi(s) = \begin{cases} E_\pi \left\{ \sum_{n=0}^{N-1} \gamma^n r_{n+k+1} \Big| s_k = s \right\} & \text{if discounted return} \\ \lim_{N\to\infty} \frac{1}{N} E_\pi \left\{ \sum_{n=0}^{N-1} r_{n+k+1} \Big| s_k = s \right\} & \text{if average return} \end{cases} \tag{3-12}$$

Similarly, the state-action value function, denoted by $Q^\pi(s, a)$, is the expected return when starting from s_k, taking action a_k and thereafter following policy π.

$$Q^\pi(s,a) = \begin{cases} E_\pi \left\{ \sum_{n=0}^{N-1} \gamma^n r_{n+k+1} \Big| s_k = s, a_k = a \right\} & \text{if discounted return} \\ \lim_{N\to\infty} \frac{1}{N} E_\pi \left\{ \sum_{n=0}^{N-1} r_{n+k+1} \Big| s_k = s, a_k = a \right\} & \text{if average return} \end{cases} \tag{3-13}$$

The relation between the state value function and the state-action value function is given by

$$V^\pi(s) = \max_a Q^\pi(s,a) \tag{3-14}$$

An optimal policy π^* is defined as a policy that maximizes both the state value function and the state-action value function.

$$\pi^* = \operatorname{argmax}_\pi V^\pi(s) \tag{3-15}$$

[3] In this thesis an MDP is assumed.

The state value function corresponding to the optimal policy is the optimal state value function and is given by

$$V^*(s) = \max_\pi V^\pi(s) \text{ for all } s \in \mathbb{S} \tag{3-16}$$

and the optimal state-action value function is given by

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \text{ for all } s \in \mathbb{S} \text{ and } a \in \mathbb{A} \tag{3-17}$$

Bellman equations If the environment can be described as an MDP it is possible to analytically calculate the optimal policy, by solving the value functions as presented in Eq. (3-12) and Eq. (3-13). In order to do this the value functions are rewritten for the optimal policy, creating the Bellman optimality equations[4]

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_s + \gamma V^*(s')\right] \tag{3-18}$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_s + \gamma \max_{a'} Q^*(s',a')\right] \tag{3-19}$$

3-2 Actor-Critic methods

Barto et al. (1983) defined for the first time a clear actor-critic structure (although the actor and critic where not named as such yet), where the actor represents the policy followed. The critic represents the estimated value function, and its role is to *criticize* the actions taken by the actor, hence the name. There are three main categories in which RL methods can be divided; critic-only methods, actor-only methods and Actor-Critic (AC) methods (Konda and Tsitsiklis, 1999). Critic-only methods learn state-action value functions and determine a policy exclusively based on the estimated value functions (e.g. Q-learning and SARSA) (Sutton and Barto, 1998). Actor-only methods on the other hand, use parametrized policies to directly estimate the gradient of the return, with respect to the parameters of the actor, and update these parameters in a direction of improvement without using value functions (e.g. see Kohl and Stone (2004)). AC methods bridge the gap between critic-only and actor-only methods.

AC methods are a form of policy gradient methods, which are widely used RL methods (e.g. Konda and Tsitsiklis (1999); Sutton et al. (2000); Kohl and Stone (2004); Tedrake et al. (2004); Matsubara et al. (2006); Peters and Schaal (2008a)). Policy gradient methods use a parametrized policy, denoted by $\pi^{\boldsymbol{\vartheta}}$, where $\boldsymbol{\vartheta} \in \mathbb{R}^m$ is the policy parameter vector. The policy must be differentiable with respect to its parameters. An

[4] The majority of the theory presented in this chapter is for the discounted return setting. Therefore only the Bellman equations for this section are given, not for the average return setting.

optimal policy parameter vector is found by applying gradient descent methods. The gradient of a function $f(\boldsymbol{\psi})$ is given by

$$\nabla_{\boldsymbol{\psi}} f(\psi_1, \ldots, \psi_m) = \begin{bmatrix} \frac{\partial f}{\partial \psi_1}(\psi_1, \ldots, \psi_m) \\ \vdots \\ \frac{\partial f}{\partial \psi_m}(\psi_1, \ldots, \psi_m) \end{bmatrix} \tag{3-20}$$

where $\psi_j, j = 1, \ldots, m$ denotes the parameters.

A gradient descent method, which is an iterative method, uses the gradient to update its parameter vector at every step, until it finds the minimum of function f. If the function f is convex, a gradient descent method is capable of finding the global minimum, otherwise only the convergence to a local minimum is guaranteed. The general update rule of the parameter vector is given by

$$\boldsymbol{\psi}_{k+1} = \boldsymbol{\psi}_k - \alpha \nabla_{\boldsymbol{\psi}} f_k(\boldsymbol{\psi}) \tag{3-21}$$

where α is the step-size parameter.

In case of RL methods the goal is to maximize the return, e.g. the discounted return or the average return (see Section 3-1). Thus, the function f in Eq. (3-21) is replaced by the return J_k, which now depends on the parametrized policy $\pi^{\boldsymbol{\vartheta}}$. Furthermore, Eq. (3-21) is concerned with finding the minimum of function f instead of the maximum of function f. Therefore, the "$-$"-sign needs to be replaced by a "$+$"-sign. This leads to the general update rule of the policy parameter vector

$$\boldsymbol{\vartheta}_{k+1} = \boldsymbol{\vartheta}_k + \alpha_a \nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) \tag{3-22}$$

where $\alpha_a \in [0, 1]$ denotes the learning rate of the actor[5]. The general policy gradient method is given in Algorithm 3-1.

Algorithm 3-1 General policy gradient method

Input: $\alpha_a, \pi^{\boldsymbol{\vartheta}}(a|s)$

Initialization: $\boldsymbol{\vartheta}_0 \leftarrow \boldsymbol{\vartheta}$

1: **for** $k = 1, 2, \ldots$ **do**
2: Take action $a_k \sim \pi^{\boldsymbol{\vartheta}}(a_k|s_k)$, observe r_{k+1}, s_{k+1}
3: Estimate gradient $\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta})$
4: $\boldsymbol{\vartheta}_k \leftarrow \boldsymbol{\vartheta}_{k-1} + \alpha_a \nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta})$
5: When gradient update $\boldsymbol{\vartheta}_k$ converged, terminate update process
6: **end for**

There are different ways of estimating the gradient $\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta})$. For example by using finite difference methods (e.g. see Kohl and Stone (2004)), which are all actor-only

[5] A slight abuse of notation is introduced here as the return is actually a function of the policy, hence $J_k(\pi^{\boldsymbol{\vartheta}})$. However, as this abuse of notation is common practice in literature it will be done here too.

methods. However, theoretical analysis and empirical evaluations have shown that actor-only methods suffer from a high variance in their gradient estimate (Bhatnagar et al., 2009b). This causes these methods to converge slowly and making them very sample inefficient. It is therefore essential to reduce the variance. One of the ways of reducing the variance is by using a value function to estimate the gradient of the return (Sigaud and Buffet, 2010). In other words, by introducing a critic.

As the exact value function is not known for large, continuous state spaces, the value function must be approximated. Function approximation is used to generalize the information gained from a small group of visited states to a much larger set of states, of which most of them never have been visited (Sutton and Barto, 1998). Approximating the value function is done by using a set of basis functions $\boldsymbol{\phi}(s)$, and the approximation itself is parametrized by a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^N$. Often, this approximation is chosen to be linear in its parameters, which greatly simplifies the calculation of the gradient with respect to its parameter vector. In that case the approximated value function is given by

$$V^{\boldsymbol{\theta}}(s) = \boldsymbol{\phi}(s)^T \boldsymbol{\theta} \tag{3-23}$$

where T denotes the transpose of a vector[6]. The gradient of the approximated value function is then given by

$$\nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s) = \boldsymbol{\phi}(s) \tag{3-24}$$

where $\nabla_{\boldsymbol{\theta}}$ denotes the gradient with respect to the parameter vector $\boldsymbol{\theta}$.

There are several ways of defining the basis functions. Some commonly used examples are neural networks and Radial Basis Functions (Buşoniu et al., 2010). Obviously, the number of parameters is smaller than the number of states; changing one parameter will lead to a change of the value function of multiple states.

Intuitively, the approximated value function should approximate the actual value function, denoted by $V^{\pi}(s)$, as well as possible[7]. A measure that is commonly used for the closeness of the approximated value function to the actual value function is the Mean Squared Error (MSE). The MSE is given by (Sutton and Barto, 1998)

$$\text{MSE}(\boldsymbol{\theta}) = \int_{\mathbb{S}} P(s)[V^{\pi}(s) - V^{\boldsymbol{\theta}}(s)]^2 ds \tag{3-25}$$

[6] Here, the superscript $\boldsymbol{\theta}$ has replaced π to explicitly denote the dependency of the function on a parameter set, which is in line with the notation used throughout this thesis. However, the value function still depends on policy π followed.

[7] In this section the state value function is used, which is stricter than the state-action value function as it does not consider the specific action taken in a state; $V^{\pi}(s)$ can be derived from $Q^{\pi}(s,a)$, but not the other way around. This removes an extra dimension from the problem compared to using $Q^{\pi}(s)$. This can be an advantage in real-life applications as states need to be visited less often to be able to construct a reliable approximation of the value function. Therefore, in literature often $V^{\pi}(s)$ is used instead of $Q^{\pi}(s,a)$.

where $P(s)$ is the probability distribution that weights the errors[8]. If it is assumed that $P(s)$ is the probability distribution of states from which samples are obtained, it can be replaced by $d^\pi(s)$ (Sutton et al., 2000). This is a state probability distribution under policy π, i.e. the chance of visiting state s following policy π. In case of the discounted return the state distribution is given by

$$d^\pi(s) = \sum_{k=0}^{\infty} \gamma^k P(s_k = s | s_0, \pi^{\boldsymbol{\vartheta}}) \tag{3-26}$$

where γ is the discount factor and P denotes a probability. The state probability distribution for the average return case is given by

$$d^\pi(s) = \lim_{k \to \infty} P(s_k = s | s_0, \pi^{\boldsymbol{\vartheta}}) \tag{3-27}$$

The latter is a stationary distribution of states and is independent of the starting state s_0 for all policies, in contrast to the discounted state distribution which depends on starting state s_0.

Analogously to the policy parameter update, gradient descent methods can be used to determine the parameter vector $\boldsymbol{\theta}$ for which the MSE is as small as possible. The gradient of the MSE is given by

$$\begin{aligned} \nabla \text{MSE}(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int_{\mathbb{S}} d^\pi(s)[V^\pi(s) - V^{\boldsymbol{\theta}}(s)]^2 ds \\ &= -2 \int_{\mathbb{S}} d^\pi(s)[V^\pi(s) - V^{\boldsymbol{\theta}}(s)] \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s) ds \end{aligned} \tag{3-28}$$

Furthermore, if assumed that the error is not minimized over all the states present in the system but only over the observed samples, which is common in real-life systems, the integral is dropped and the state distribution reduces to a constant value. Implementing Eq. (3-28) in Eq. (3-21) gives the update rule of the parameter vector of the value function

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_c [V^\pi(s_k) - V^{\boldsymbol{\theta}}(s_k)] \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s_k) \tag{3-29}$$

where $\alpha_c \in [0, 1]$ denotes the learning rate of the critic.

However, as the actual value function $V^\pi(s_k)$ is not known, it is replaced by an unbiased estimate of the value function, denoted by $\hat{V}^\pi(s_k)$. For this unbiased estimate it holds that $E\{\hat{V}^\pi(s_k)\} = V^\pi(s_k)$. The part in between squared brackets in Eq. (3-29) then becomes

[8] In this section continuous state and action spaces are assumed. However, Sutton and Barto (1998) use discrete state spaces. It was therefore necessary to replace the summation over all states with an integral over the complete state space.

$$\hat{V}^{\pi}(s_k) - V^{\boldsymbol{\theta}}(s_k) \tag{3-30}$$

and is the so called Temporal Difference (TD) error. This error is the driving force of learning in both the actor and the critic. The exact form of the TD error depends on the return used, e.g. the discounted return or the average return.

Sutton and Barto (1998) derived the exact form of the TD error using the discounted return. As said before, the actual value function is not known, and is therefore replaced by an unbiased estimate. Ideally, the unbiased estimate should equal the actual value function. By solving the Bellman equations as presented in Section 3-1, it is possible to express the unbiased estimate in terms of the reward and the value function of the next state, which gives

$$\delta_k = r_{k+1} + \gamma V^{\boldsymbol{\theta}}(s_{k+1}) - V^{\boldsymbol{\theta}}(s_k) \tag{3-31}$$

and is known as the TD error for the discounted return.

In e.g. Konda and Tsitsiklis (1999) and Bhatnagar et al. (2007) the TD error using the average return is derived, analogously to the TD error of the discounted return, and is given by

$$\delta_k = r_{k+1} - J_k(\boldsymbol{\vartheta}) + V^{\boldsymbol{\theta}}(s_{k+1}) - V^{\boldsymbol{\theta}}(s_k) \tag{3-32}$$

The TD error is used to evaluate the action taken. If the error is positive, action a_k taken in state s_k led to a higher reward than expected and therefore this action should be selected more often in the future when that particular state is accounted. If the error is negative that action should be selected less often in the future. This relation between actor updates and critic updates, was e.g. derived in Bhatnagar et al. (2009b), where the gradient of the return $J_k(\boldsymbol{\vartheta})$ is written as

$$\begin{aligned} \nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) &= \delta_k \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a_k|s_k) \\ &= \delta_k \frac{\nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a_k|s_k)}{\pi^{\boldsymbol{\vartheta}}(a_k|s_k)} \end{aligned} \tag{3-33}$$

The general form of AC methods, using an approximated value function and gradient descent-methods to obtain the optimal parameters, is described in Algorithm 3-2. A graphical representation of the AC structure is given in Figure 3-2.

3-2-1 episodic Natural Actor-Critic

The eNAC method is first proposed by Peters et al. (2003) and subsequently formalized in a number of papers (Peters et al., 2005; Peters and Schaal, 2006, 2007; Peters et al., 2007; Peters and Schaal, 2008a,b). It is a model-free method, but the most important property of the method is that it does not require the explicit definition of the basis

Algorithm 3-2 General Actor-Critic method

Input: $\gamma, \alpha_c, \alpha_a, \pi^{\boldsymbol{\vartheta}}(a|s), V^{\boldsymbol{\theta}}(s)$
Initialization: $\boldsymbol{\vartheta}_0 \leftarrow \boldsymbol{\vartheta}, \boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}$

1: **for** $k = 1, 2, \ldots$ **do**
2: Take action $a_k \sim \pi^{\boldsymbol{\vartheta}}(a_k|s_k)$, observe r_{k+1}, s_{k+1}
3: $\delta_k = \hat{V}^{\pi}(s_k) - V^{\boldsymbol{\theta}}(s_k)$
4: Calculate gradient $\nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s_k)$
5: $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} + \alpha_c \delta_k \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s_k)$
6: $\boldsymbol{\vartheta}_k \leftarrow \boldsymbol{\vartheta}_{k-1} + \alpha_a \delta_k \frac{\nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a_k|s_k)}{\pi^{\boldsymbol{\vartheta}}(a_k|s_k)}$
7: When gradient update $\boldsymbol{\vartheta}_k$ converged, terminate update process
8: **end for**

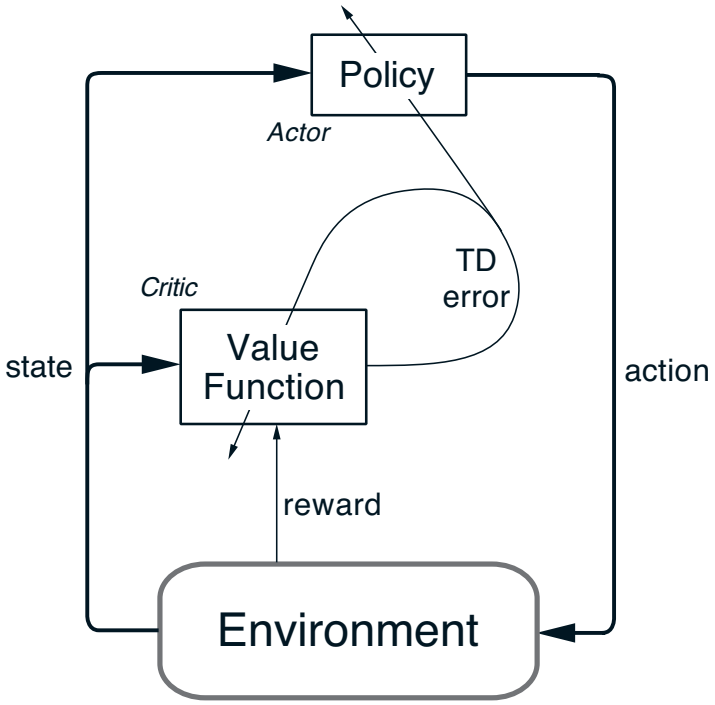


Figure 3-2: The general Actor-Critic structure. The figure is adopted from Sutton and Barto (1998).

functions of the value function approximation. This makes it a particularly suitable method to apply to systems with a high dimensional state space, in which it is hard to define a good set of basis functions. The eNAC method is successfully applied by Peters and Schaal (2008b) to a system consisting of 14 states and 7 actions, each continuous, with a policy parameter vector of length 70. The system was a robotic arm consisting of seven joints holding a baseball bat, which task it was to hit a baseball placed on a T-stick such that flies away as far as possible. Furthermore, successful applications have been reported by e.g. Kim et al. (2010) in the task of robot-environment contact.

The natural gradient As the name implies the eNAC method makes use of the natural policy gradient, instead of the normal policy gradient, to update the policy parameters. The natural gradient is shown to reduce the variance of the gradient estimate, and thus improve speed and convergence, compared to the normal gradient (Bhatnagar et al., 2009a). In the next section the difference between the normal gradient and natural gradient is explained, and some of the advantages of using natural gradients are listed.

The steepest ascent direction is defined as the vector $\Delta\boldsymbol{\psi}$ that maximizes (Kakade, 2001)

$$\max_{\Delta\boldsymbol{\psi}} J_k(\boldsymbol{\psi} + \Delta\boldsymbol{\psi}), \qquad \text{s.t. } ||\Delta\boldsymbol{\psi}||^2 \leq \varepsilon \tag{3-34}$$

where it is required that $||\Delta\boldsymbol{\psi}||^2$ is held to a small constant (Kakade, 2001). This requirement can be written in matrix form, giving

$$||\Delta\boldsymbol{\psi}||^2 = \Delta\boldsymbol{\psi}^T G(\boldsymbol{\psi})\Delta\boldsymbol{\psi} \tag{3-35}$$

The steepest ascent direction is then given by $G^{-1}(\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}} J_k(\boldsymbol{\psi})$. For the normal gradient the matrix $G(\boldsymbol{\psi})$ is the identity matrix. However, by using the identity matrix the requirement $||\Delta\boldsymbol{\psi}||^2$ is different for every parametrization of the policy. To overcome this problem Amari (1998) suggested to use the Fisher information matrix, instead of the identity matrix, for $G(\boldsymbol{\psi})$. The general form of this matrix is given by (Peters et al., 2003)

$$F(\boldsymbol{\psi}) = \int_{\mathbb{X}} p(\boldsymbol{x}) \nabla_{\boldsymbol{\psi}} \log p(\boldsymbol{x}) \nabla_{\boldsymbol{\psi}} \log p(\boldsymbol{x})^T dx \tag{3-36}$$

where $p(\boldsymbol{x})$ is a probability distribution function for variable $\boldsymbol{x}$ and $\boldsymbol{\psi}$ denotes a parameter vector.

By introducing the Fisher information matrix the gradient update no longer depends on the parametrization of the policy (Amari, 1998), i.e. the Fisher information matrix is invariant of the policy parametrization as the distance between two points will be the

same regardless of the choice of parametrization (Kakade, 2001)[9]. The natural gradient in the AC setting is now given by

$$\tilde{\nabla}_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) = F^{-1}(\boldsymbol{\vartheta}) \nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) \tag{3-37}$$

where $F(\boldsymbol{\vartheta})$ denotes the Fisher information matrix and $\boldsymbol{\vartheta}$ the policy parameters.

Some of the important properties of natural gradients are:

- The natural gradient is shown to reduce the variance of the gradient estimate, and thus improve speed and convergence, compared to the normal gradient (Bhatnagar et al., 2009a).
- Convergence to the local minimum is guaranteed (Amari, 1998).
- The natural gradient is independent of the policy parametrization (Amari, 1998).

In order to derive the eNAC method the return $J_k(\boldsymbol{\vartheta})$ as used in Section 3-2 is rewritten to obtain (Sutton et al., 2000)

$$J_k(\boldsymbol{\vartheta}) = \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \pi^{\boldsymbol{\vartheta}}(a|s) \mathcal{R}^a_{ss'} da ds \tag{3-38}$$

where $d^{\pi}(s)$ as defined in Eq. (3-26) or Eq. (3-27) and $\mathcal{R}^a_s$ as defined in Eq. (3-11).

In Sutton et al. (2000) the policy gradient theorem is defined, in which the gradient of the return with respect to the policy parameters is given by

$$\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) = \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) Q^{\pi}(s,a) da ds \tag{3-39}$$

Furthermore, in Sutton et al. (2000) it is proven that the state-action value function can be replaced by a parametrized function approximation without affecting the unbiasedness of the gradient estimate. For convenience this proof is repeated here. Let $f^{\boldsymbol{w}}(s,a)$ be an approximation to $Q^{\pi}(s,a)$, parametrized by a parameter vector $\boldsymbol{w}$. By making use of the parameter update rule presented in Eq. (3-21), it is found that if the parameter update process is converged to a local optimum the following equation holds.

$$\int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \pi^{\boldsymbol{\vartheta}}(a|s) [Q^{\pi}(s,a) - f^{\boldsymbol{w}}(s,a)] \nabla_w f^{\boldsymbol{w}}(s,a) da ds = 0 \tag{3-40}$$

[9] Consider a point in Cartesian coordinates and the same point in polar coordinates, i.e. two different parametrizations of the same point. Changing the Cartesian coordinates by a certain small norm will generally result in a different point than when changing the polar coordinates by the same small norm, due to the sin and cos present in the conversion from Cartesian to polar coordinates.

Theorem 3. *If $f^{\boldsymbol{w}}(s,a)$ is chosen such that*

$$\nabla_w f^{\boldsymbol{w}}(s,a) = \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) \frac{1}{\pi^{\boldsymbol{\vartheta}}(a|s)} = \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a|s) \tag{3-41}$$

it can be shown that

$$\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) = \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) f^{\boldsymbol{w}}(s,a) dads \tag{3-42}$$

Proof. By combining Eq. (3-40) and Eq. (3-41)

$$\int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) [Q^{\pi}(s,a) - f^{\boldsymbol{w}}(s,a)] dads = 0 \tag{3-43}$$

is obtained. Because the expression above equals zero it is possible to subtract it from the policy gradient theorem as presented in Eq. (3-39), resulting in

$$\begin{aligned} \nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) &= \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) Q^{\pi}(s,a) dads \\ &\quad - \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) [Q^{\pi}(s,a) - f^{\boldsymbol{w}}(s,a)] dads \\ &= \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) [Q^{\pi}(s,a) - Q^{\pi}(s,a) + f^{\boldsymbol{w}}(s,a)] dads \\ &= \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) f^{\boldsymbol{w}}(s,a) dads \end{aligned} \tag{3-44}$$

□

Following from Eq. (3-41) the function approximation $f^{\boldsymbol{w}}(s,a)$ is named the compatible function approximation, as it must be compatible with the policy parametrization. If $f^{\boldsymbol{w}}(s,a)$ is assumed to be linear in its parameters (see Eq. (3-23)), it is given by

$$f^{\boldsymbol{w}}(s,a) = \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(s,a)^T \boldsymbol{w} \tag{3-45}$$

where $\boldsymbol{w}$ are the parameters of the compatible function approximation and it holds that $\dim \boldsymbol{\vartheta} = \dim \boldsymbol{w}$.

Now one step back is taken to Eq. (3-39), and a baseline $b^{\pi}(s)$ is introduced, which is an arbitrary function of state s. This baseline is added to Eq. (3-39) in order to obtain

$$\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) = \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) \left[Q^{\pi}(s,a) - b^{\pi}(s) \right] dads \tag{3-46}$$

Adding a baseline to this function can be done as it does not introduce a bias, which can be shown by

$$\begin{aligned}\int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) b^{\pi}(s) dads &= \int_{\mathbb{S}} d^{\pi}(s) b^{\pi}(s) \nabla_{\boldsymbol{\vartheta}} \left(\int_{\mathbb{A}} \pi^{\boldsymbol{\vartheta}}(a|s) da \right) ds \\ &= \int_{\mathbb{S}} d^{\pi}(s) b^{\pi}(s) \nabla_{\boldsymbol{\vartheta}}(1) ds \\ &= 0\end{aligned} \tag{3-47}$$

However, the baseline can be used to minimize the variance of the gradient estimate as is proven in e.g. Greensmith et al. (2004), where the optimal baseline, i.e. the baseline that reduces the variance the most, is given by $V^{\pi}(s)$. This transforms Eq. (3-46) into

$$\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) = \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) \left[Q^{\pi}(s,a) - V^{\pi}(s) \right] dads \tag{3-48}$$

where

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{3-49}$$

is known as the advantage function (Baird, 1994). The advantage function gives the advantage of choosing action a over the average performance in state s; it is zero when the action was optimal and negative for any sub-optimal action.

If the result of Eq. (3-45) is examined more closely, it is found that the compatible function approximation is zero-mean with respect to the action distribution, thus

$$\begin{aligned}\int_{\mathbb{A}} \pi^{\boldsymbol{\vartheta}}(a|s) f^{\boldsymbol{w}}(s,a) da &= \boldsymbol{w}^T \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) da \\ &= 0, \qquad \forall s \in \mathbb{S}\end{aligned} \tag{3-50}$$

as it follows from Eq. (3-47) that $\int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \pi^{\boldsymbol{\vartheta}}(a|s) da = 0$. The compatible function approximation should not be seen as the approximation of $Q^{\pi}(s,a)$ but as an approximation of the advantage function as presented in Eq. (3-49). The convergence, as used in Eq. (3-40), does not require that $f^{\boldsymbol{w}}(s,a)$ finds the correct absolute value of the actions in each state, but rather that it finds the correct relative value of the actions in each state (Sutton et al., 2000).

By combining Eq. (3-41), Eq. (3-45) and Eq. (3-48) it is possible to write the gradient of the expected return with respect to the policy parameters as

$$\begin{aligned}\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) &= \int_{\mathbb{S}} d^{\pi}(s) \int_{\mathbb{A}} \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a|s) \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a|s)^T \boldsymbol{w} dads \\ &= G(\boldsymbol{\vartheta}) \boldsymbol{w}\end{aligned} \tag{3-51}$$

Due to the fact that $\pi^{\boldsymbol{\vartheta}}(a|s)$ is a function chosen by the user it is possible to evaluate the integral over the actions present in Eq. (3-51), often analytically or at least empirically,

without performing all actions. However, the state distribution $d^\pi(s)$ is not known, which requires the use of episodes[10] to estimate $G(\boldsymbol{\vartheta})$.

An important result from Peters et al. (2003) is that matrix $G(\boldsymbol{\vartheta})$ equals the Fisher information matrix, for both the discounted as well as the average return setting. The natural gradient update is thus given by

$$\begin{aligned} \tilde{\nabla}_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta}) &= F^{-1}(\boldsymbol{\vartheta}) G(\boldsymbol{\vartheta}) \boldsymbol{w} \\ &= \boldsymbol{w} \end{aligned} \tag{3-52}$$

This result reduces the data needed to get a good estimate of $\nabla_{\boldsymbol{\vartheta}} J_k(\boldsymbol{\vartheta})$ drastically, as normally much more data is necessary in order to get a good estimate of $G(\boldsymbol{\vartheta})$ than for a good estimate of $\boldsymbol{w}$ (estimating a vector versus estimating a matrix).

It is possible to write the Bellman equation of the state-action value function (see Eq. (3-19)) in terms of the advantage function and the state value function, which gives[11]

$$\begin{aligned} Q^\pi(s,a) &= A^\pi(s,a) + V^\pi(s) \\ &= r(s,a) + \gamma \int_{\mathbb{S}} \mathcal{P}^a_{ss'} V^\pi(s') ds \end{aligned} \tag{3-53}$$

where $\mathcal{P}^a_{ss'}$ as defined in Eq. (3-10).

Now assume that a set of samples (s_k, a_k, r_k, s_{k+1}) is given. Furthermore, if the advantage function is replaced by the compatible function approximation of Eq. (3-45) and an appropriate approximation of the value function is chosen (see Eq. (3-23)) a set of linear equations is obtained

$$\nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a_k|s_k)^T \boldsymbol{w} \approx r(s_k, a_k) + \gamma \boldsymbol{\phi}(s_{k+1})^T \boldsymbol{\theta} - \boldsymbol{\phi}(s_k)^T \boldsymbol{\theta} \tag{3-54}$$

Algorithms that make use of Eq. (3-54) are referred to as Natural Actor-Critic (NAC) algorithms (Peters and Schaal, 2008b). Notice the similarity between the right-hand side of the equation and the TD error as presented in Eq. (3-31).

However, this expression relies on the set of basis functions chosen to approximate the state value function. This can be problematic in high dimensional state spaces, where it is difficult to define a good set of basis functions. Therefore, episodes are introduced. Eq. (3-54) can then be summed up over the episode and the following equation is obtained

$$\sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a_k|s_k)^T \boldsymbol{w} = \sum_{k=0}^{N-1} \gamma^k r(s_k, a_k) + \gamma^N \boldsymbol{\phi}(s_N)^T \boldsymbol{\theta} - \boldsymbol{\phi}(s_0)^T \boldsymbol{\theta} \tag{3-55}$$

[10] An episode is a sequence of states and actions; $(s_0, a_0, r_0, \ldots, s_{N-1}, a_{N-1}, r_{N-1}, s_N)$. In literature different terms are used to denote an episode, such as a roll-out or trajectory.

[11] The equations below are all given for the discounted return setting. However, the equations for the average return setting can simply be obtained by solving the Bellman equations for that setting, inserting them in Eq. (3-53) and follow the same derivation from there.

where N is the number of samples present in a single episode.

The term $\gamma^N \boldsymbol{\phi}(s_N)^T \boldsymbol{\theta}$ disappears for discounted learning when $N \to \infty$ or for episodic tasks where the value of the terminal state is zero. If furthermore a single start state (or a zero-mean start state distribution) is assumed only one additional value is necessary to estimate $\boldsymbol{\phi}(s_0)^T \boldsymbol{\theta}$. This is equal to estimating the value of the critic at the start state s_0, and the basis function used is given by $\phi(s_0) = 1$. The problem of Eq. (3-55) is now reduced to a simple regression problem of the form

$$\sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a_k|s_k)^T \boldsymbol{w} + \theta = \sum_{k=0}^{N-1} \gamma^k r(s_k, a_k) \tag{3-56}$$

with dim $\boldsymbol{\vartheta} + 1$ unknowns.

This equation can be written into matrix form

$$\begin{bmatrix} \sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a_k|s_k)^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \theta \end{bmatrix} = \sum_{k=0}^{N-1} \gamma^k r(s_k, a_k) \tag{3-57}$$

In order to solve this problem use is made of linear regression techniques. Eq. (3-57) is therefore written as

$$\begin{bmatrix} \boldsymbol{w} \\ \theta \end{bmatrix} = (\boldsymbol{\Psi}^T \boldsymbol{\Psi})^{-1} \boldsymbol{\Psi}^T \boldsymbol{R} \tag{3-58}$$

with

$$\begin{aligned} \boldsymbol{\Psi} &= \begin{bmatrix} \sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a_k|s_k)^T & 1 \end{bmatrix} \\ \boldsymbol{R} &= \sum_{k=0}^{N-1} \gamma^k r(s_k, a_k) \end{aligned} \tag{3-59}$$

However, in order to obtain a good gradient of estimate $\boldsymbol{w}$ multiple episodes are needed. Therefore, the system in Eq. (3-58) is rewritten to

$$\begin{bmatrix} \boldsymbol{w}_e \\ \theta_e \end{bmatrix} = \left(\sum_{i=1}^{e} \boldsymbol{\Psi}_i^T \boldsymbol{\Psi}_i \right)^{-1} \left(\sum_{i=1}^{e} \boldsymbol{\Psi}_i^T \boldsymbol{R}_i \right) \tag{3-60}$$

where e denotes the episode number.

For simplicity introduce

$$\begin{aligned} \boldsymbol{M}_e &= \sum_{i=1}^{e} \boldsymbol{\Psi}_i^T \boldsymbol{\Psi}_i \\ \boldsymbol{b}_e &= \sum_{i=1}^{e} \boldsymbol{\Psi}_i^T \boldsymbol{R}_i \end{aligned} \tag{3-61}$$

The system of Eq. (3-60) can finally be written as

$$\begin{bmatrix} \boldsymbol{w}_e \\ \theta_e \end{bmatrix} = \boldsymbol{M}_e^{-1} \boldsymbol{b}_e \tag{3-62}$$

The pseudocode describing the eNAC method is given in Algorithm 3-3 and in Figure 3-3 a flow chart of the algorithm is shown[12]. A clear distinction must be made between episodes and the actual update of the policy parameter vector. In every update step u a number of episodes, denoted by e, is performed. After a sufficiently number of episodes the gradient estimate $\boldsymbol{w}$ is converged and the policy parameter vector $\boldsymbol{\vartheta}$ is updated. After a number of updates the policy parameter vector $\boldsymbol{\vartheta}$ will converge and the optimization is terminated.

The computational cost of this method is linear in the number of value function parameters $\boldsymbol{w}$ and θ and quadratic in the number of policy parameters $\boldsymbol{\vartheta}$ as follows from Eq. (3-62).

Algorithm 3-3 episodic Natural Actor-Critic method

Input: $\alpha_a, \pi^{\boldsymbol{\vartheta}}(a|s), p(s_0)$
Initialization: $\boldsymbol{\vartheta}_0 \leftarrow \boldsymbol{\vartheta}, \boldsymbol{w}_0 \leftarrow \boldsymbol{0}, \theta_0 \leftarrow 0, \boldsymbol{M}_0 \leftarrow \boldsymbol{0}, \boldsymbol{b}_0 \leftarrow \boldsymbol{0}$

for $u = 1, 2, \ldots$ **do**
  for $e = 1, 2, \ldots$ **do**
    Draw initial state $s_0 \sim p(s_0)$
    for $k = 0, \ldots, N-1$ **do**
      Take action $a_k \sim \pi^{\boldsymbol{\vartheta}}(a_k|s_k)$, observe r_{k+1}, s_{k+1}
    end for
    Determine $\boldsymbol{\Psi}_e, \boldsymbol{R}_e$ using Eq. (3-59)
    Update $\boldsymbol{M}_e = \boldsymbol{M}_{e-1} + \boldsymbol{\Psi}_e^T \boldsymbol{\Psi}_e$ and $\boldsymbol{b}_e = \boldsymbol{b}_{e-1} + \boldsymbol{\Psi}_e^T \boldsymbol{R}_e$
    $[\boldsymbol{w}_e, \theta_e]^T = \boldsymbol{M}_e^{-1} \boldsymbol{b}_e$
    Until gradient estimate $\boldsymbol{w}_e$ converged
  end for
  Update policy parameter vector: $\boldsymbol{\vartheta}_u \leftarrow \boldsymbol{\vartheta}_{u-1} + \alpha_a \boldsymbol{w}_e$
  When gradient update $\boldsymbol{\vartheta}_u$ converged, terminate update process
end for

3-3 Conclusions

After having discussed the modelling of a robot and the gait generation in the previous chapter, this chapter was completely dedicated to the learning method used in this thesis, the eNAC method. This method is solidly anchored in the theory of RL, which

[12] The algorithm presented in Algorithm 3-3 is slightly different from the algorithm presented in Peters et al. (2003); here a clear distinction is made between parameter updates while in Peters et al. (2003) this is not done. The reason it is done here is for better understandability.

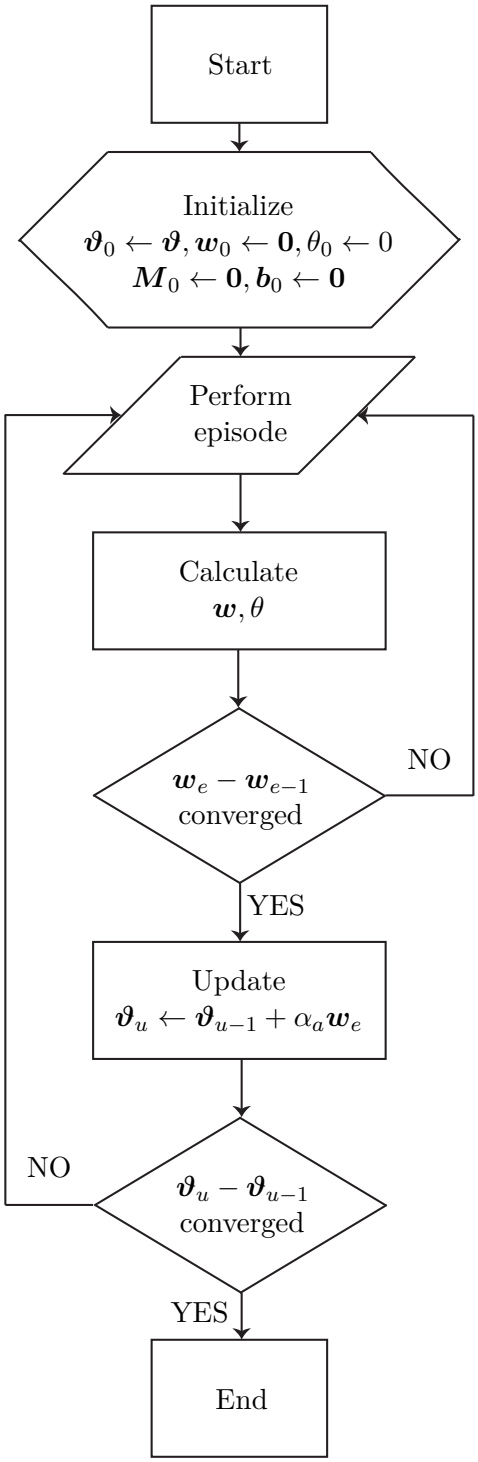


Figure 3-3: This figure shows a flowchart of the eNAC method, used to visualize Algorithm 3-3. A clear distinction must be made between episodes and the actual update of the policy parameter vector. In every update step u a number of episodes, denoted by e, is performed. After a sufficiently number of episodes the gradient estimate $\boldsymbol{w}$ is converged and the policy parameter vector $\boldsymbol{\vartheta}$ is updated. After a number of updates the policy parameter vector $\boldsymbol{\vartheta}$ will converge and the optimization is terminated.

is a learning framework inspired by how animals learn to deal with new situations, using a trial-and-error approach. Some important properties of eNAC are that it is a model-free method and it is capable of handling systems with large (continuous) state spaces. The model as constructed in Chapter 2 is thus not a requirement of the optimization method, but is merely used to accurately determine the response of the system to the actions performed, i.e. it represents the state transition function. The methods capability of handling large state spaces is largely achieved by not requiring the explicit definition of basis functions of the value function approximation. Furthermore, the episodic nature of the method makes it a suitable method to combine with legged locomotion as episodes can be clearly defined, e.g. by defining a number of steps for the robot to take. The structure of the learning problem, how to learn the optimal gait parameters using the eNAC method, is discussed in Chapter 4.

Chapter 4

A learning model

This chapter deals with the implementation of the theory as presented in Chapter 2 and Chapter 3 into the problem of legged locomotion optimization of a six-legged hexapod robot. The first part discussed is how to frame the problem to be implemented in the Reinforcement Learning (RL) architecture. Decisions have to be made on the states present in the state signal, the actions applied to the system and the reward functions used to judge the behaviour of the system. Second, the implementation of the episodic Natural Actor-Critic (eNAC) method is discussed. Finally, the parameter values of the ground contact model and the reference trajectory tracker are defined. The implementation of the Max-Plus gait generation is not discussed in this chapter as it follows directly from the theory as presented in Section 2-2 and no decisions regarding the implementation have to be made. The Matlab functions used to implement the Max-Plus gait generation are given in Appendix B-1.

4-1 Framing the complete problem

In this section the complete problem is framed such that it can be incorporated into the RL structure: what are the states, what are the actions and how are the rewards computed. In order to derive the problem in a systematic way, use is made of a modified version of the control structure of the Switching Max-Plus-linear model (Section 2-2-2).

In this thesis optimization within a certain gait is considered, such that the supervisory control block can be dropped. Furthermore, disturbances in the leg trajectories, such as a leg that is being hold in place, are not considered making it possible to drop the feedback loop to the Max-Plus gait scheduler. The control structure that remains is shown in Figure 4-1. The block diagram is arranged from high-level control at the left of the diagram, to low-level control at the right of the diagram. Furthermore, the decision is made to implement PD-controllers as the reference trajectory trackers.

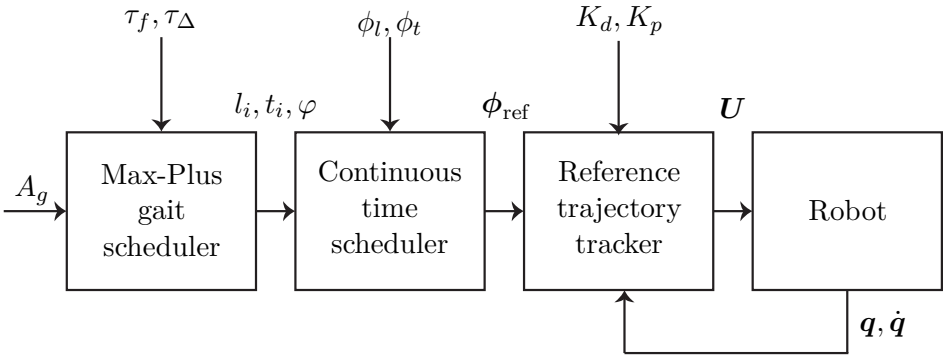


Figure 4-1: The modified version of the control structure as presented in Section 2-2-2. The first block on the left is the Max-Plus gait scheduler parametrized by τ_f and τ_Δ. The gait scheduler generates the lift-off and touchdown event times of the legs, l_i and t_i, based on the matrix A_g containing the structure of the gait. Furthermore, it outputs the phase φ of the Max-Plus gait generation. Second, there is the continuous time scheduler parametrized by ϕ_l and ϕ_t, which transforms the discrete events to continuous time actions. Third, there is a reference trajectory tracker, parametrized by K_p and K_p, which task it is to make sure the legs follow the reference trajectory as accurate as possible. The last block present is the robot itself. The signal going from the reference trajectory tracker to the robot is a control signal for the DC motors driving the legs, and is denoted by $\boldsymbol{U}$. Furthermore, there is a feedback loop present between the robot and the reference trajectory controller feeding back state variables $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.

The assumption is made that all legs are identical and can share the same controller parameters.

The problem is derived as a set of equations, each equation denoted by f_j, $j = 1, \ldots, 4$, parametrized by some parameters. The highest-level equation is denoted by f_1 and the lowest-level equation by f_4. At the far left side of the block diagram in Figure 4-1 the Max-Plus gait scheduler is located. Based on the type of gait used and the parameters of the flight time of the legs and the double stance time of the legs, the lift-off and touchdown event times are generated and the phase of the Max-Plus gait generation is outputted (a more comprehensive explanation is given in Section 2-2). This can be denoted by

$$[l_i, t_i, \varphi]^T = f_1(\tau_f, \tau_\Delta, A_g), \qquad i = 1, \ldots, 6 \tag{4-1}$$

where l_i are the timings of the lift-off events of the legs, t_i are timings of the touchdown events of the legs, φ is the phase of the Max-Plus gait generation, τ_f is the flight time of the legs, τ_Δ is the double stance time of the legs, A_g is the gait matrix containing the structure of the gait and T denotes the transpose of a vector.

The second block considered is the continuous time scheduler, which generates a reference trajectory for each leg to follow based on the phase of the Max-Plus gait generation, the event times and two parameters; the lift-off and touchdown angles. The function describing the continuous time scheduler is given by

$$\boldsymbol{\phi}_{\text{ref}} = f_2(\varphi, l_i, t_i, \phi_l, \phi_t), \qquad i = 1, \ldots, 6 \tag{4-2}$$

where $\boldsymbol{\phi}_{\text{ref}}$ is a vector of length six containing the reference trajectories for the legs to follow, ϕ_l is the lift-off angle and ϕ_t is the touchdown angle.

The third block considered is the reference trajectory tracker, which outputs the voltages that have to be applied to the DC motors in order for the legs to follow their reference trajectories. PD-controllers are used as reference trajectory tracking controllers, parametrized by two parameters: K_d and K_p.

$$\boldsymbol{U} = f_3(\dot{\boldsymbol{\phi}}, \boldsymbol{\phi}, \boldsymbol{\phi}_{\text{ref}}, K_d, K_p) \tag{4-3}$$

where $\boldsymbol{U}$ is a vector of length six containing the voltages that are supplied to the DC motors driving the legs, $\dot{\boldsymbol{\phi}}$ is a vector of length six containing the phase velocities of the legs, $\boldsymbol{\phi}$ is a vector of length six containing the phases of the legs.

Finally, the last block outputs the derivative of the internal state of the robot based on the input $\boldsymbol{U}$ and the internal state $\boldsymbol{x}$ of the robot. The equation describing this last block is given by

$$\dot{\boldsymbol{x}} = f_4(\boldsymbol{x}, \boldsymbol{U}) \tag{4-4}$$

where $\boldsymbol{x} = [\boldsymbol{q}^T, \dot{\boldsymbol{q}}^T]^T$ is the internal state vector of the robot and $\boldsymbol{q}$ are the internal states of the robot (Section 2-1). It is important to notice that for recirculating legs, as assumed in this thesis, the leg phases and phase velocities $\boldsymbol{\phi}$ and $\dot{\boldsymbol{\phi}}$, as presented in Eq. (4-3), are equal to the leg angles and angular velocities as present in the equations of motion.

If Eq. (4-1)-Eq. (4-3) are then substituted in Eq. (4-4) the following function, describing the complete problem, is found.

$$\begin{aligned} \dot{\boldsymbol{x}} &= f_4(\boldsymbol{x}, f_3(\dot{\boldsymbol{\phi}}, \boldsymbol{\phi}, f_2(\varphi, f_1(\tau_f, \tau_\Delta, A_g), \phi_l, \phi_t), K_d, K_p)) \\ &= \bar{f}_4(\boldsymbol{q}, \dot{\boldsymbol{q}}, \varphi, \boldsymbol{\psi}) \end{aligned} \tag{4-5}$$

where

$$\boldsymbol{\psi} = [K_d, K_p, \phi_l, \phi_t, \tau_f, \tau_\Delta]^T \tag{4-6}$$

is a parameter vector containing the parameters of the problem[1].

The parameter vector $\boldsymbol{\psi}$ is of interest to the problem, as this vector contains the parameters that can be chosen by the user, and thus optimized. The lowest-level parameters are the parameters of the reference trajectory tracking controller, K_d and K_p. These parameters are straightforward to optimize using standard PD control tools, and will therefore not be included in the optimization process. This leaves four parameters,

[1] In this thesis only a single gait is assumed within an optimization, and A_g is therefore not considered as a parameter that can change, but as a fixed parameter. It is thus not included in the parameter vector $\boldsymbol{\psi}$.

which qualify for automatic optimization. The main difference between τ_f, τ_Δ and ϕ_l, ϕ_t is that the latter depend directly on the state of the robot, as is explained in Section 2-2-2. Therefore, these parameters require fast updating, i.e. within the cycle time of the legs. The former parameters τ_f and τ_Δ, which are the highest-level parameters, do not depend on the state of the robot and therefore do not require fast updating.

Now the structure of the problem is known, it is possible to cast it in the form of a RL problem, i.e. with an environment and agent, and reward, state and action signals between those two parts. As the parameters ϕ_l and ϕ_t depend directly on the states, it is chosen to select these two parameters as the actions, which will be optimized using the eNAC method. The parameters K_d, K_p, τ_f and τ_Δ can then be optimized within the environment using "traditional" optimization techniques. It must be emphasized that in this thesis the latter four parameters are not optimized, but are fixed during experiments.

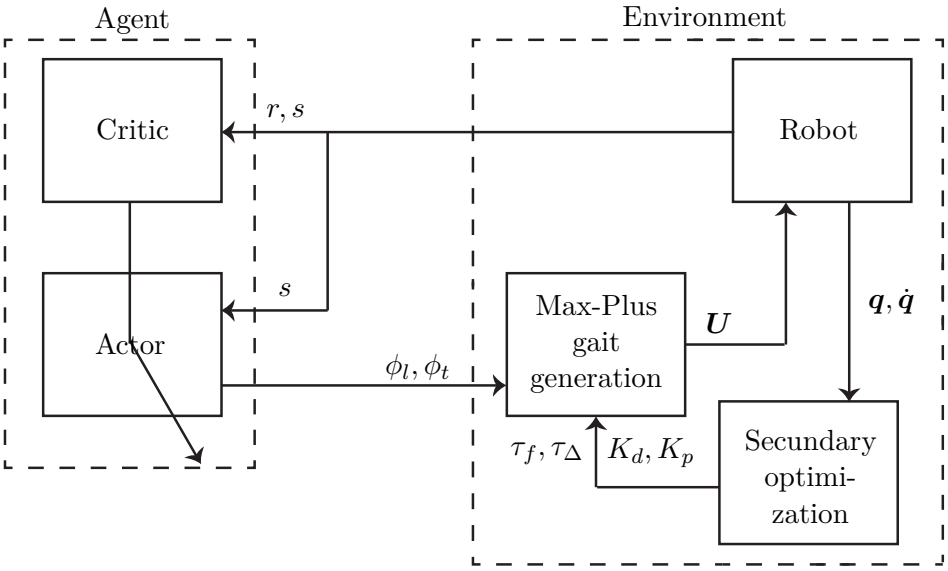


Figure 4-2: Framing the problem in the RL structure. This figure shows how the problem is build up. As is required in RL problems, there are an agent and an environment present. Between the environment and the agent information exchange in the form of state signal s, reward signal r and actions ϕ_l, ϕ_t, takes place. The Max-Plus block contains the Max-Plus gait scheduler, the continuous time scheduler and the reference trajectory tracker (see Figure 4-1), and outputs a vector $\boldsymbol{U}$ containing the voltages for the DC motors. Within the environment a second optimization method is present, which optimizes the parameters τ_f, τ_Δ, K_p and K_d based on the internal states $\dot{\boldsymbol{q}}, \boldsymbol{q}$ of the robot. These parameters are optimized outside of the agent as they do not depend on the dynamics of the system.

4-1-1 States

The state $s_k \in \mathbb{S}$, where $\mathbb{S}$ is the set of possible states and k denotes the current time step, is defined by a signal from the environment to the agent, representing (some) properties of the environment (Sutton and Barto, 1998).

By combining Eq. (4-5) with the model of the robot, as derived in Section 2-1-1, it is possible to formulate the complete state vector of the system. This vector is given by

$$\begin{aligned} s_k &= [\boldsymbol{q}^T, \dot{\boldsymbol{q}}^T, \varphi]^T \\ &= \left[x, z, \beta, \boldsymbol{\phi}^T, \dot{x}, \dot{z}, \dot{\beta}, \dot{\boldsymbol{\phi}}^T, \varphi\right]^T \end{aligned} \tag{4-7}$$

where x is the position of the robot in x-direction, z is the position of the robot in z-direction, β is the angle of the body, $\boldsymbol{\phi}$ is a vector of length six containing the angles of the legs and φ is the phase of the Max-Plus gait generation. The total state vector contains 19 states.

However, such a large amount of states can still be problematic, although the eNAC method is successfully applied to problems with over 10 states (e.g. Peters and Schaal (2008b)). The number of parameters in the policy is related to the number of states present in the system, where it holds that the number of parameters is greater than the number of states. By decreasing the number of states, the complexity of the problem can be greatly reduced.

State reduction

In Section 3-1 the definition of the Markov property is given, and is repeated here: "when the environment is described by a state signal that summarizes all past sensations compactly, yet in such a way that all relevant information is preserved, the state signal is said to have the Markov property". Thus, a state can be removed from the state vector if that state, at a previous moment in time, is not necessary to describe the current state of the robot, i.e. when reducing the state vector the underlying dynamics of the mechanical system must be kept in mind in order to ensure that the Markov property is not violated. In Figure 4-3 it can be seen that the position $\boldsymbol{q}$ directly depends on the velocity $\dot{\boldsymbol{q}}$, and the velocity directly depends on the acceleration $\ddot{\boldsymbol{q}}$. Hence, states can only be removed, from right to left in the diagram, if it can be proven that the force F does not depend on the removed state(s)[2].

The x-position The first reduction of the state vector is obtained by removing the x-position. The removal of x is done as the position of the robot is not of interest to the problem; the robot does not have to navigate to a certain point or follow a specific trajectory. It follows directly from the equations of motion derived in Section 2-1 (see also Appendix A) that the dynamics do not depend on the state x, as in these equations the state x is not present[3].

The reduced state vector is given by

[2] This thus includes also the external forces, which could depend on the state of the system.

[3] This includes the reaction forces acting on the robot. Although the ground contact model at first sight requires the x-position of the robot, it actually only needs the relative change in x-position and not the absolute x-position. The relative change can be inferred from the velocity and therefore the external forces do not depend on the state x.

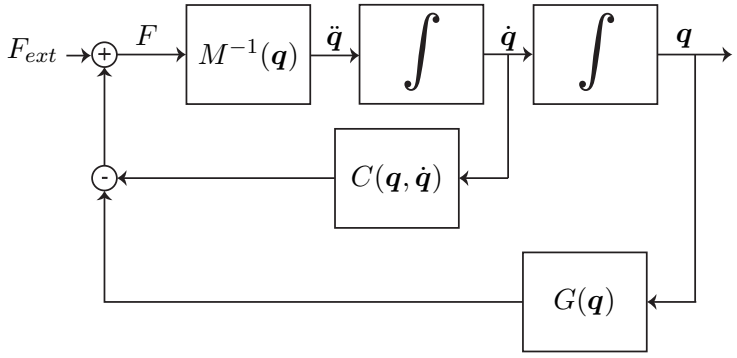


Figure 4-3: The underlying dynamics of (mechanical) systems. This figure shows that the position $\boldsymbol{q}$ directly depends on the velocity $\dot{\boldsymbol{q}}$, and the velocity directly depends on the acceleration $\ddot{\boldsymbol{q}}$. Hence, states can only be removed, from right to left, if it can be proven that the force F does not depend on the removed state(s).

$$s_k = \left[z, \beta, \boldsymbol{\phi}^T, \dot{x}, \dot{z}, \dot{\beta}, \dot{\boldsymbol{\phi}}^T, \varphi\right]^T \tag{4-8}$$

where $\boldsymbol{\phi}$ and $\dot{\boldsymbol{\phi}}$ are vectors of length six, and contains 18 states.

Virtual leg The robot has a total of six legs, contributing a total of 12 states to the state vector in the form of $\boldsymbol{\phi}$ and $\dot{\boldsymbol{\phi}}$. A large reduction can thus be potentially obtained by eliminating one or more legs from the state vector. However, as follows from Section 2-1-1 the leg angles ϕ_i and angular velocities $\dot{\phi}_i$ are present in the equations of motion. Hence, they cannot be removed from the state vector without violating the Markov property. Nevertheless, it is believed that a reduction can be obtained by introducing the notion of a "virtual leg" (Holmes et al., 2006). This virtual leg is constructed by averaging the six leg angles and leg angular velocities to get a single, virtual leg angle $\bar{\phi}$ and a single, virtual leg angular velocity $\dot{\bar{\phi}}$. Basically, the problem is simplified from a robot with six legs to a robot with a single leg, which does satisfy the Markov property[4].

The further reduced state vector is given by

$$s_k = \left[z, \beta, \bar{\phi}, \dot{x}, \dot{z}, \dot{\beta}, \dot{\bar{\phi}}, \varphi\right]^T \tag{4-9}$$

and contains 8 states.

State vector

The (final) reduced state vector is given in Eq. (4-9) and contains 8 states. In this section the state space is determined.

[4] As is explained in Section 4-1, the reference trajectory controller is not a part of the agent but part of the environment. Therefore, it has direct access to the leg angles and leg angular velocities needed to control the position of the legs. Introducing a virtual leg thus not influence the quality of the reference tracking control.

- z: the height of body of the robot is continuous on $\mathbb{R}$. In practice, however, the height is limited by the ground on one side.
- β: the pitch angle of the body is continuous on $\mathcal{S}^1$. In practice it is not possible for the pitch angle to reach the end of its range as this means the robot turned upside down. Therefore, β is considered to be continuous in an interval of $\mathbb{R}$.
- $\bar{\phi}$: the virtual leg position is continuous on $\mathcal{S}^1$.
- $\dot{x}$: the velocity of the robot in x-direction is continuous on $\mathbb{R}$.
- $\dot{z}$: the velocity of the robot in z-direction is continuous on $\mathbb{R}$.
- $\dot{\beta}$: the angular velocity of the body of the robot is continuous on $\mathbb{R}$.
- $\dot{\bar{\phi}}$: the angular velocity of the virtual leg is continuous on $\mathbb{R}$.
- φ: the phase of the Max-Plus gait generation is continuous on $\mathcal{S}^1$.

It is thus found that the state vector is divided into two parts; one containing six states that are continuous in an interval of $\mathbb{R}$ and the second containing two states that are continuous on $\mathcal{S}^1$. The state space of the state vector is given by $\mathbb{R}^6 \times \mathbb{T}^2 \to \mathbb{S}$.

4-1-2 Actions

The action $a_k \in \mathbb{A}(s_k)$, where $\mathbb{A}(s_k)$ is the set of actions available in state s_k, is the output of the agent (Sutton and Barto, 1998).

From Figure 4-2 it follows that the output of the agent are the lift-off and touchdown angles, which are the inputs to the continuous time scheduler (see Section 2-2-2). The action vector is given by

$$a_k = [\phi_l, \phi_t]^T \tag{4-10}$$

where ϕ_l is the lift-off angle of the legs and ϕ_t the touchdown angle of the legs.

For both actions it holds that they are continuous on $\mathcal{S}^1$, and can be projected onto a line: $\mathcal{S}^1 \to \mathbb{R}$. Thus, the action space is given by $\mathbb{T}^2 \to \mathbb{A}$ where both actions can take on values in the range $[-\pi, \pi]$.

4-1-3 Rewards

A reward function, $\rho : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, determines the reward the agent receives at each time step k. The function must be shaped such that the robot optimizes the desired objective, by maximizing the return (Sutton and Barto, 1998).

It is therefore necessary to define which objectives are considered before a reward function can be defined.As explained in Section 1-1, animals have optimized their gait with

respect to power consumption at a certain velocity. Therefore, the focus of this thesis will be on maximizing the velocity in x-direction and minimizing the power consumption. However, these two objectives are conflicting with each other. Intuitively it can be said that maximizing the velocity will negatively influence the power consumption and minimizing the power consumption will negatively influence the velocity. When defining a reward function it is thus necessary to incorporate some kind of trade-off between the two objectives. This trade-off is in this thesis incorporated in two different ways, giving two types of reward functions: a reward function based on a weighted average of the velocity and power consumption, and reward functions based on the specific resistance. In the experiments a choice can be made which reward function to use. First, the weighted average reward function is discussed and after that the specific resistance reward functions are given.

Weighted average

To combine maximizing $\dot{x}$ and minimizing P in an optimal way, reward functions of both objectives are first defined separately and normalized in the range $[0, 1]$. As a result, the reward functions are dimensionless. The advantage of defining the reward function as a weighted average is that it is a linear combination of multiple objectives and, if necessary, the weight of one objective can be set to 0, isolating a single objective.

Velocity A reward function capable of accomplishing the maximization of $\dot{x}$ is given by

$$\rho_{\dot{x}} = \frac{\dot{x} - \dot{x}_{\min}}{\dot{x}_{\max} - \dot{x}_{\min}} \tag{4-11}$$

where $\dot{x}$ is the velocity of the robot in x-direction, $\dot{x}_{\min}$ is the minimum velocity and $\dot{x}_{\max}$ is the maximum velocity the robot can achieve.

The minimum velocity of the robot is equal to 0 m/s; it is thus assumed the robot will only walk in the forward direction. The maximum velocity is not trivial, as this is the maximum speed the robot can achieve, which is exactly what is being optimized. To circumvent this problem a guess must be made of the maximum velocity. This guess is based on the configuration of the robot in combination with the gait generation method. It is believed that the forward velocity of the robot can be approximated by (Lopes et al., 2009)

$$\dot{x} \approx l_{\text{leg}} \frac{(\phi_l - \phi_t)}{\tau_g} \tag{4-12}$$

where l_{leg} is the leg length, ϕ_l the lift-off angle, ϕ_t the touchdown angle and τ_g the ground time of the legs.

As given in Section 4-3, the length of a leg is 0.15 m, and assuming a maximum difference between ϕ_l and ϕ_t of π rad, this equation transforms to

$$\dot{x}_{\max} \approx 0.15 \, \frac{m}{rad} \frac{\pi \; rad}{\tau_g} \approx \frac{0.5 \; m}{\tau_g} \tag{4-13}$$

In practice, this maximum velocity will not be exceeded.

Power consumption The second objective is to minimize the total power consumption of the robot, where the total power consumption consists of the power consumed by the six DC motors driving the legs plus the power consumed by the on-board electronics. A reward function capable of accomplishing this objective is given by

$$\rho_P = \frac{P_{\max} - P}{P_{\max} - P_{\min}} \tag{4-14}$$

where P is the power consumption of the robot, $P_{\min}$ is the minimum power consumption and $P_{\max}$ the maximum power consumption.

The minimum power consumption is determined using Zebro. It is found that the minimum power consumption of the robot when switched on, but not moving, is given by $P_{\min} = 24 \; W$. The maximum power consumption of each individual DC motor is $60 \; W$, giving $P_{\max} = 384 \; W$ (see for more details on the motors used Section 4-3).

Combining velocity and power consumption Combining Eq. (4-11) and Eq. (4-14), gives the weighted average reward function

$$\begin{aligned} \rho_{\text{wa}} &= \alpha \rho_{\dot{x}} + (1 - \alpha) \rho_P \\ &= \alpha \left(\frac{\dot{x} - \dot{x}_{\min}}{\dot{x}_{\max} - \dot{x}_{\min}} \right) + (1 - \alpha) \left(\frac{P_{\max} - P}{P_{\max} - P_{\min}} \right) \end{aligned} \tag{4-15}$$

where α is the weight of the velocity and $(1-\alpha)$ is the weight of the power consumption. Furthermore, it holds that

$$0 \leq \alpha \leq 1 \tag{4-16}$$

The choice of α depends on which objective is emphasized and is thus an open parameter in the experiments.

To complete the reward function a negative reward is given to the agent when the body touches the ground. This negative reward is set equal to -1, bounding the rewards in the interval $[-1, 1]$. The total weighted average reward function becomes

$$\rho_{\text{wa}} = \begin{cases} \alpha \left(\dfrac{\dot{x} - \dot{x}_{\min}}{\dot{x}_{\max} - \dot{x}_{\min}} \right) + (1 - \alpha) \left(\dfrac{P_{\max} - P}{P_{\max} - P_{\min}} \right) & \text{if run completed successfully} \\ -1 & \text{if the body touches the ground} \end{cases} \tag{4-17}$$

The values of the parameters present in Eq. (4-17) are given in Table 4-1.

Table 4-1: Parameters of the reward function

Parameter	Value
$\dot{x}_{\min}$	$0\ m/s$
$\dot{x}_{\max}$	$0.5/\tau_g\ m/s$
$P_{\min}$	$24\ W$
$P_{\max}$	$384\ W$

Specific resistance

A second way of combining the velocity and power consumption into a single reward function is by making use of the specific resistance (Section 1-1). Additionally, to emphasize the importance of the velocity it is possible to define the so-called speed-weighted specific resistance, which is given by

$$f_{\text{swsr}} = \frac{P}{mg\dot{x}^3} \qquad [1/m^2] \tag{4-18}$$

where P is the power consumption, m is the mass of the robot, g the acceleration due to gravity and x the velocity.

As can be seen this equation is not dimensionless any longer, making it difficult to compare the result of the optimization to other robots.

The (speed-weighted) specific resistance is a cost function, i.e. it must be minimized. However, a reward function needs to be defined in such a way that the better the behaviour the higher the reward. A way of transforming a cost function into a reward function is by considering the negative of the cost function. For the specific resistance the reward function becomes

$$\rho_{\text{sr}} = -\frac{P}{mg\dot{x}} \tag{4-19}$$

and for the speed-weighted specific resistance the reward function becomes

$$\rho_{\text{swsr}} = -\frac{P}{mg\dot{x}^3} \qquad [1/m^2] \tag{4-20}$$

Analogously to Eq. (4-17), a negative reward is given to the agent if the body of the robot touches the ground. However, as the reward functions in Eq. (4-19) and Eq. (4-20) are not bounded to a specific interval, it is more difficult to determine a negative reward that influences the total reward sufficiently. E.g. substituting the maximum power consumption following from Table 4-1 in both equations and letting $\dot{x} \to \dot{x}_{\min}$ shows that the minimal reward goes to $-\infty$, but substituting $P_{\min}$ and $\dot{x}_{\max}$

shows that the maximal reward $\approx 0^-$. Therefore, the negative reward is defined by a large, negative value, -100 and -10000, respectively.

The complete reward functions now become

$$\rho_{\text{sr}} = \begin{cases} -\dfrac{P}{mg\dot{x}} & \text{if run completed successfully} \\ -100 & \text{if the body touches the ground} \end{cases} \tag{4-21}$$

and

$$\rho_{\text{swsr}} = \begin{cases} -\dfrac{P}{mg\dot{x}^3} & \text{if run completed successfully} \\ -10000 & \text{if the body touches the ground} \end{cases} \tag{4-22}$$

4-2 Application of the eNAC method

Although the theory behind the eNAC method has been presented in Section 3-2-1, it cannot be applied directly to the problem as described in Section 4-1. Some additional properties need to be defined first. In Section 4-2-1 the policy used is constructed in two steps and in Section 4-2-2 the stopping criteria for a parameter update and the optimization are defined.

4-2-1 The policy

A formal definition of the policy is given in Section 3-1, where the policy is defined as: "a stochastic policy, denoted by $\pi(a_k = a|s_k = s)$, gives the probability that $a_k = a$ if $s_k = s$".

In this section two different policies are considered: a deterministic policy and a stochastic policy. The reason behind this is that, as explained in section Section 3-2-1, the eNAC method requires a stochastic policy in order to perform gradient updates. It is, however, simplest to first derive a deterministic policy and then transform this into a stochastic policy. In order to distinguish the two policies $\mu(s)$ is used to denote the deterministic policy[5] and $\pi(a|s)$ is used to denote the stochastic policy. Although the eNAC method is a model-free RL method and does not require a model of the environment, knowledge about the environment is used to construct a policy.

[5] As the chance of selecting an action in a certain state equals 1 with a deterministic policy, the notation of the deterministic policy is simplified from $\mu(a|s)$ to $\mu(s)$ throughout this thesis.

4-2-1-1 The deterministic policy

As is stated in Section 4-1-2, two actions are used in the system, the lift-off and touch-down angles ϕ_l and ϕ_t, respectively. Hence, a policy consists of two parts, one outputting ϕ_l and the other outputting ϕ_t. First, the deterministic policy $\mu(s)$ is considered. Each part is represented by a deterministic function of the states (a total of 8 states is considered, see Eq. (4-9)), giving only a single output. This function must capture the correct dependency of the action on all states. Furthermore, in Section 4-1-1 it is concluded that a part of the states is continuous on $[-\pi, \pi] \in \mathbb{R}$ and another part is continuous on $\mathcal{S}^1$. The non-periodic states present in the state vector of Eq. (4-9) are $z, \beta, \dot{x}, \dot{z}, \dot{\beta}$ and $\dot{\bar{\phi}}$, and the periodic states present are $\bar{\phi}$ and φ (for a complete explanation of each symbol the reader is referred to Section 4-1-1). In order to capture both types of states correctly in the deterministic functions it is decided to form each function from several sub-functions, each describing a different type of state. In the following paragraphs three types of sub-functions are considered; polynomial functions, Fourier functions and bilinear functions. Although there is (almost) no knowledge of the actual shape of the deterministic function, it is believed that these sub-functions are capable of capturing enough information on the state dependency of the actions to construct high quality deterministic functions. The deterministic functions for each action are given by

$$\begin{aligned} \mu_{\phi_l}(s) &= \phi_l \\ \mu_{\phi_t}(s) &= \phi_t \end{aligned} \tag{4-23}$$

Polynomial function Polynomial functions are used to approximate the non-periodic state influence on the action. A general representation of an n^{th} order polynomial function is given by (Stewart, 2003)

$$f_p(s_{np}) = a_n s_{np}^n + a_{n-1} s_{np}^{n-1} + \ldots + a_1 s_{np} + a_0 \tag{4-24}$$

where $a_0, a_1, a_2, \ldots, a_{n-1}, a_n$ are the parameters of the polynomial and s_{np} denotes the non-periodic states present in the system.

For each non-periodic state variable one polynomial function is necessary, replacing x in Eq. (4-24) with the state variable considered. The number of parameters introduced per state variable is given by $(n+1)$, where n is the order of the polynomial approximation. E.g. using a 3^{rd} order polynomial for each state will introduce $4 \times 6 = 24$ parameters.

Fourier function To approximate the periodic states present in the state vector Fourier functions are used. Fourier functions have the ability to approximate periodic functions, e.g. a sawtooth or square wave signal, by a linear combination of sin and cos functions. The Fourier approximation of a function $f(x)$, periodic on the interval $[-L, L]$, is given by (Boyce and DiPrima, 2005)

$$f_f(s_p) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi s_p}{L}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi s_p}{L}\right) \tag{4-25}$$

with L a half of the period of the function $f(x)$ that is approximated, n the order of the Fourier approximation, s_p denotes the periodic states present in the system and

$$\begin{aligned} a_0 &= \frac{1}{L}\int_{-L}^{L} f(s_p) ds_p \\ a_n &= \frac{1}{L}\int_{-L}^{L} f(s_p) \cos\left(\frac{n\pi s_p}{L}\right) ds_p \\ b_n &= \frac{1}{L}\int_{-L}^{L} f(s_p) \sin\left(\frac{n\pi s_p}{L}\right) ds_p \end{aligned} \tag{4-26}$$

are the parameters of the Fourier function.

However, the exact shape of the periodic function $f(s_p)$ is not known as this is the unknown function that is being approximated. Therefore, the exact values of the parameters a_0, a_n, b_n cannot be calculated but have to be found using the optimization method. The number of parameters introduced per state variable is given by $(2n+1)$, where n is the order of the Fourier approximation. What remains is the variable L, which is half of the period of the approximated function. However, it is known that the period for both states is equal and given by the cycle time of a leg λ (see Section 2-2-2). Thus, $L = \lambda/2$.

Bilinear function The third function, a bilinear function[6], introduces a relationship between different state variables. The function is built up from three parts: one part relating the non-periodic states to each other, one part relating the periodic states to each other and a third part relating the non-periodic states to the periodic states. A bilinear combination between non-periodic states can be given by

$$f_{b,1}(s_{np}) = \begin{cases} a_{i,j} s_{np,i} s_{np,j} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{4-27}$$

where a denotes the parameters. The total number of parameters is given by $(n^2 - n)$, where n is the number of non-periodic state variables. However, this introduces all expressions twice, as e.g. $s_{np,1}s_{np,2} = s_{np,2}s_{np,1}$, and it is therefore possible to drop half of the expressions, leaving $((n^2 - n)/2)$ parameters.

The combination between periodic states can be given by

[6] This name is not entirely correct, as a combination between periodic states, or between periodic and non-periodic states, is not bilinear, due to the Fourier approximation. However, for convenience all combinations are denoted as bilinear.

$$f_{b,2}(s_p) = \begin{cases} \sum_{n=0}^{\infty} \left[a_{i,1} \cos\left(\frac{n\pi s_{p,i}}{L}\right) + a_{i,2} \sin\left(\frac{n\pi s_{p,i}}{L}\right) \right] \\ \sum_{l=0}^{\infty} \left[a_{j,1} \cos\left(\frac{l\pi s_{p,j}}{L}\right) + a_{j,2} \sin\left(\frac{l\pi s_{p,j}}{L}\right) \right] & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{4-28}$$

where n, l are the order of approximation of each function. The bilinear function has a total of $(4m^2 - 4m)$ parameters (a multiplication of parameters can be replaced by a single parameter), where m is the number of periodic state variables. Again, all expressions are introduced twice and thus half of the expressions can be dropped, leaving $(2m^2 - 2m)$ parameters.

Finally, the combination between periodic and non-periodic states can be done in a similar fashion.

$$f_{b,3}(s_p, s_{np}) = s_{np,i} \left(a_{i,j,1} \cos\left(\frac{\pi s_{p,j}}{L}\right) + a_{i,j,2} \sin\left(\frac{\pi s_{p,j}}{L}\right) \right), \qquad \forall i, j \tag{4-29}$$

This function introduces a total of $2nm$ variables, where n is the number of non-periodic states and m is the number of periodic states.

Constant The polynomial part and the Fourier part each introduce one parameter per state variable that acts as an offset, i.e. a parameter that is not multiplied by a state value. It is therefore possible to lump all these parameters together into a single parameter. Doing this reduces the number of parameters present in each deterministic function by $(n + m - 1)$, where n is the number of non-periodic states and m is the number of periodic states.

The deterministic functions as presented in Eq. (4-23) are now rewritten as a combination of the above presented sub-functions.

$$\begin{aligned} \mu_{\phi_l}(s) &= f_p(s_{np}) + f_f(s_p) + f_{b,1}(s_{np}) + f_{b,2}(s_p) + f_{b,3}(s_p, s_{np}) + c_{\phi_l} \\ \mu_{\phi_t}(s) &= f_p(s_{np}) + f_f(s_p) + f_{b,1}(s_{np}) + f_{b,2}(s_p) + f_{b,3}(s_p, s_{np}) + c_{\phi_t} \end{aligned} \tag{4-30}$$

As follows from the previous paragraphs, each function part, and thus the total function, is parametrized by a set of parameters. The parameter vector of the function $\mu_{\phi_l}(s)$ is given by $\boldsymbol{\nu}_l$ and the parameter vector of the function $\mu_{\phi_t}(s)$ is given by $\boldsymbol{\nu}_t$. The functions are from now on denoted by $\mu_{\phi_l}^{\boldsymbol{\nu}_l}(s)$ and $\mu_{\phi_t}^{\boldsymbol{\nu}_t}(s)$, respectively, in order to emphasize their dependency on a certain parameter set. Combining the two deterministic functions gives the deterministic policy

$$\mu^{\boldsymbol{\nu}}(s) = [\mu_{\phi_l}^{\boldsymbol{\nu}_l}(s), \mu_{\phi_t}^{\boldsymbol{\nu}_t}(s)]^T \tag{4-31}$$

where T denotes the transpose of a vector.

The decisions on the order of the polynomial functions, the order of the Fourier functions and whether or not to include the bilinear part in the deterministic policy depends on the experiments performed and are therefore not fixed.

4-2-1-2 The stochastic policy

In this thesis the stochastic policy is obtained by perturbing the values of ϕ_l and ϕ_t, following from their deterministic functions $\mu_{\phi_l}^{\boldsymbol{\nu}_l}(s)$ and $\mu_{\phi_t}^{\boldsymbol{\nu}_t}(s)$, by adding a value from a normal distribution $\epsilon \sim \mathcal{N}(0, \sigma^2)$ to them[7]. This is a common way of defining a stochastic policy, utilized in many papers (Matsubara et al., 2006; Nakamura et al., 2007; Peters and Schaal, 2008b). The standard deviations can be different for each action, and are denoted by σ_{ϕ_l} and σ_{ϕ_t}, respectively. The stochastic actions applied to the system are then given by

$$\begin{aligned} \hat{\mu}_{\phi_l}^{\boldsymbol{\nu}_l}(s) &= \mu_{\phi_l}^{\boldsymbol{\nu}_l}(s) + \epsilon_{\phi_l} \\ \hat{\mu}_{\phi_t}^{\boldsymbol{\nu}_t}(s) &= \mu_{\phi_t}^{\boldsymbol{\nu}_t}(s) + \epsilon_{\phi_t} \end{aligned} \tag{4-32}$$

with $\epsilon_{\phi_l} \sim \mathcal{N}(0, \sigma_{\phi_l}^2)$ and $\epsilon_{\phi_t} \sim \mathcal{N}(0, \sigma_{\phi_t}^2)$. It is important to notice that $\hat{\mu}_{\phi_l}^{\boldsymbol{\nu}_l}(s)$ and $\hat{\mu}_{\phi_t}^{\boldsymbol{\nu}_t}(s)$ are scalar values, while $\mu_{\phi_l}^{\boldsymbol{\nu}_l}(s)$ and $\mu_{\phi_t}^{\boldsymbol{\nu}_t}(s)$ are scalar functions of the states. To avoid confusion, the action symbol a will be used to denote the stochastic actions applied to the system, where $a = [\hat{\mu}_{\phi_l}^{\boldsymbol{\nu}_l}(s), \hat{\mu}_{\phi_t}^{\boldsymbol{\nu}_t}(s)]^T$.

As the explorations added to the deterministic policy are obtained from normal distributions, it is possible to write the stochastic policy in the form of a multivariate normal distribution (Tong, 1990), as is done in Eq. (4-33).

$$\pi^{\boldsymbol{\vartheta}}(a|s) = \frac{1}{|2\pi\Sigma|^{-1/2}} \exp^{\left(-\frac{1}{2}(a-\mu^{\boldsymbol{\nu}}(s))^T\Sigma^{-1}(a-\mu^{\boldsymbol{\nu}}(s))\right)} \tag{4-33}$$

with $a = [\hat{\mu}_{\phi_l}^{\boldsymbol{\nu}_l}(s), \hat{\mu}_{\phi_t}^{\boldsymbol{\nu}_t}(s)]^T$, $\mu^{\boldsymbol{\nu}}(s) = [\mu_{\phi_l}^{\boldsymbol{\nu}_l}(s), \mu_{\phi_t}^{\boldsymbol{\nu}_t}(s)]^T$, $\Sigma = \text{diag}(\sigma_{\phi_l}^2, \sigma_{\phi_t}^2)$ and $\boldsymbol{\vartheta}$ the parameter vector of the policy.

The parameter vector of the stochastic policy consists of the parameter sets of the deterministic policy plus the exploration parameters σ_{ϕ_l} and σ_{ϕ_t}, giving

$$\boldsymbol{\vartheta} = [\boldsymbol{\nu}_l^T, \boldsymbol{\nu}_t^T, \sigma_{\phi_l}, \sigma_{\phi_t}]^T \tag{4-34}$$

An essential part of the eNAC method is the calculation of the gradient of the logarithm of the policy with respect to the policy parameters (Section 3-2-1), which is given by $\nabla_{\boldsymbol{\vartheta}} \log \pi^{\boldsymbol{\vartheta}}(a|s)$. In order to obtain this gradient, first the logarithm of the policy is calculated in Eq. (4-35).

[7] If $\sigma = 0$ it holds that $\epsilon = 0$ and thus the stochastic policy equals the deterministic policy.

$$\begin{aligned}\log \pi^{\boldsymbol{\vartheta}}(a|s) &= \log\left[\frac{1}{|2\pi\Sigma|^{-1/2}}\exp^{\left(-\frac{1}{2}(a-\mu^{\boldsymbol{\nu}}(s))^T\Sigma^{-1}(a-\mu^{\boldsymbol{\nu}}(s))\right)}\right] \\ &= \log\left[\frac{1}{|2\pi\Sigma|^{-1/2}}\right] - \left[\frac{1}{2}(a-\mu^{\boldsymbol{\nu}}(s))^T\Sigma^{-1}(a-\mu^{\boldsymbol{\nu}}(s))\right]\end{aligned} \tag{4-35}$$

The next step is to calculate the gradient of Eq. (4-35) with respect to the policy parameter vector $\boldsymbol{\vartheta}$. The policy parameter vector consists of four parts, making it possible to calculate the gradient with respect to each policy parameter vector part separately, which is done in Eq. (4-36)-Eq. (4-39). The separate gradients are then joined together in Eq. (4-40) to form the total gradient.

$$\nabla_{\boldsymbol{\nu}_l}\log\pi^{\boldsymbol{\vartheta}}(a|s) = \left(\frac{a_{\phi_l}-\mu^{\boldsymbol{\nu}_l}_{\phi_l}(s)}{\sigma^2_{\phi_l}}\right)\nabla_{\boldsymbol{\nu}_l}\mu^{\boldsymbol{\nu}_l}_{\phi_l}(s) \tag{4-36}$$

with $a_{\phi_l} = \hat{\mu}^{\boldsymbol{\nu}_l}_{\phi_l}(s)$.

$$\nabla_{\boldsymbol{\nu}_t}\log\pi^{\boldsymbol{\vartheta}}(a|s) = \left(\frac{a_{\phi_t}-\mu^{\boldsymbol{\nu}_t}_{\phi_t}(s)}{\sigma^2_{\phi_t}}\right)\nabla_{\boldsymbol{\nu}_t}\mu^{\boldsymbol{\nu}_t}_{\phi_t}(s) \tag{4-37}$$

with $a_{\phi_t} = \hat{\mu}^{\boldsymbol{\nu}_t}_{\phi_t}(s)$.

$$\nabla_{\sigma_{\phi_l}}\log\pi^{\boldsymbol{\vartheta}}(a|s) = -\frac{1}{\sigma_{\phi_l}} + \frac{(a_{\phi_l}-\mu^{\boldsymbol{\nu}_l}_{\phi_l}(s))^2}{\sigma^3_{\phi_l}} \tag{4-38}$$

$$\nabla_{\sigma_{\phi_t}}\log\pi^{\boldsymbol{\vartheta}}(a|s) = -\frac{1}{\sigma_{\phi_t}} + \frac{(a_{\phi_t}-\mu^{\boldsymbol{\nu}_t}_{\phi_t}(s))^2}{\sigma^3_{\phi_t}} \tag{4-39}$$

The complete gradient thus becomes

$$\begin{aligned}\nabla_{\boldsymbol{\vartheta}}\log\pi^{\boldsymbol{\vartheta}}(a|s) &= \Big[(\nabla_{\boldsymbol{\nu}_l}\log\pi^{\boldsymbol{\vartheta}}(a|s))^T, (\nabla_{\boldsymbol{\nu}_t}\log\pi^{\boldsymbol{\vartheta}}(a|s))^T, ... \\ &\qquad \nabla_{\sigma_{\phi_l}}\log\pi^{\boldsymbol{\vartheta}}(a|s), \nabla_{\sigma_{\phi_t}}\log\pi^{\boldsymbol{\vartheta}}(a|s)\Big]^T \\ &= \begin{bmatrix} \left(\left(\frac{a_{\phi_l}-\mu^{\boldsymbol{\nu}_l}_{\phi_l}(s)}{\sigma^2}\right)\nabla_{\boldsymbol{\nu}_l}\mu^{\boldsymbol{\nu}_l}_{\phi_l}(s)\right)^T \\ \left(\left(\frac{a_{\phi_t}-\mu^{\boldsymbol{\nu}_t}_{\phi_t}(s)}{\sigma^2}\right)\nabla_{\boldsymbol{\nu}_t}\mu^{\boldsymbol{\nu}_t}_{\phi_t}(s)\right)^T \\ -\frac{1}{\sigma_{\phi_l}} + \frac{(a_{\phi_l}-\mu^{\boldsymbol{\nu}_l}_{\phi_l}(s))^2}{\sigma^3_{\phi_l}} \\ -\frac{1}{\sigma_{\phi_t}} + \frac{(a_{\phi_t}-\mu^{\boldsymbol{\nu}_t}_{\phi_t}(s))^2}{\sigma^3_{\phi_t}} \end{bmatrix}\end{aligned} \tag{4-40}$$

4-2-2 The stopping criteria

During the optimization two questions are of importance: when is the parameter update converged and when is the optimization itself converged? The stopping criterion of a parameter update determines if the gradient estimates found within the parameter update are converged well enough such that an update of the parameter vector can take place. The stopping criterion of the optimization determines if the parameter updates have converged well enough and the optimization can terminate.

Parameter update In case of the eNAC method it is necessary to compare subsequent parameter gradient vectors to each other (Algorithm 3-3), and check if they are converging. One way of checking the convergence of two different vectors is to calculate the angle between the vectors. This angle can be calculated with

$$\alpha = \arccos\left(\frac{\boldsymbol{w}_e^T \boldsymbol{w}_{e-1}}{||\boldsymbol{w}_e||\ ||\boldsymbol{w}_{e-1}||}\right) \tag{4-41}$$

where $\boldsymbol{w}_e$ and $\boldsymbol{w}_{e-1}$ are the vectors of which the relative angle is calculated and $||\boldsymbol{w}_e||$ and $||\boldsymbol{w}_{e-1}||$ are the lengths of the vectors.

If the angle α is below a certain threshold ε_w during a certain horizon τ_w, the stopping criterion of a parameter update is satisfied. The pseudocode of the stopping criterion of the parameter update is given in Algorithm 4-1.

Algorithm 4-1 Stopping criterion parameter update

Input: threshold ε_w, horizon τ_w

1: **if** $\angle(w_{e-\tau_w}, w_e) < \varepsilon_w$ **then**
2: parameter update converged and parameter vector updated
3: **end if**

Optimization The goal of the eNAC method is to maximize the return J_k. As this is a scalar value, it is only necessary to check if the difference between subsequent returns is below a certain threshold. Again, a threshold ε_J and horizon τ_J are defined, and the pseudocode of the resulting stopping criterion of the optimization is given in Algorithm 4-2.

Algorithm 4-2 Stopping criterion optimization

Input: threshold ε_J, horizon τ_J

1: **if** $|J_{u-\tau_J} - J_u| < \varepsilon_J$ **then**
2: optimization converged
3: **end if**

4-3 Defining the model parameter values

In Section 2-1 the equations of motion of the sagittal plane of Zebro are derived and the ground contact is modelled. Furthermore, in Section 4-1 it is established PD-controllers will be used as reference trajectory trackers. To be able to implement this information into a Matlab/Simulink model it is necessary to define the parameter values of each part of the model. First, the dimensions of the robot are listed. Second, the parameters of the DC motors driving the legs and the gearbox placed between the motor shaft and legs are given. After that the parameters of the leg-ground contact are defined and finally the parameters of the PD-controller are determined.

Dimensions of the robot

A schematic overview of the robot, as implemented in Matlab/Simulink, is given in Figure 4-4. In Table 4-2 the dimensions of the robot are given, plus the mass of the body and legs.

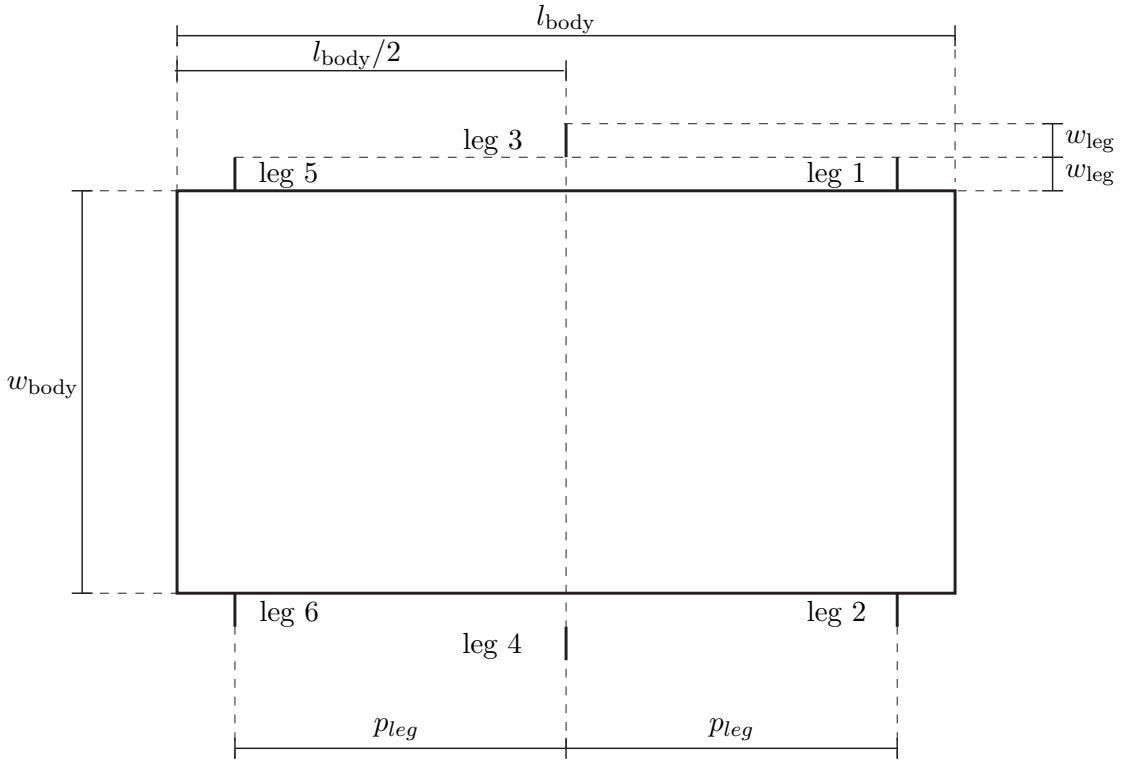
Table 4-2: Dimensions of the model of the robot

Parameter	Value
w_{body}	240 mm
l_{body}	464 mm
h_{body}	60 mm
w_{leg}	20 mm
l_{leg}	150 mm
p_{leg}	197.5 mm
m_{body}	10 kg
m_{leg}	0.1 kg

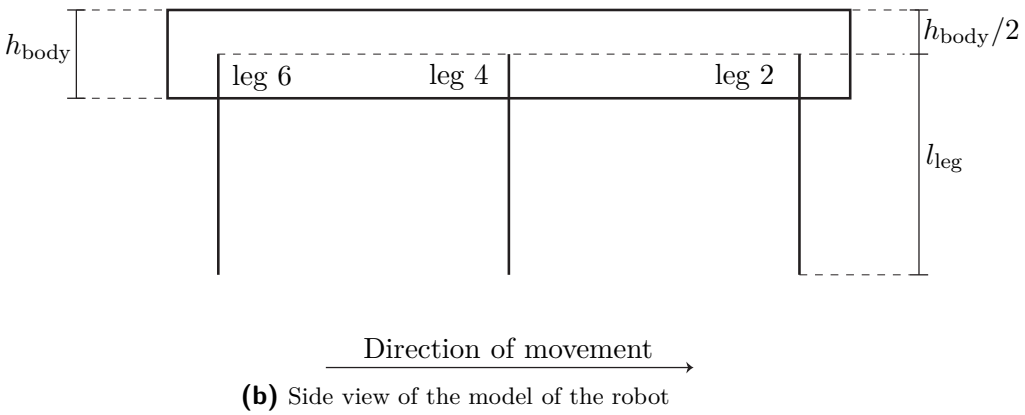
Driving the legs

The values of the motor parameters used in the model are taken from the documentation provided by the manufacturer and are given in Table 4-3. The DC motors used are: Maxon DC Motor, RE 30 Ø30 mm, Graphite Brushes, 60 W, order number 310007.

Furthermore, there is a gearbox placed between each DC motor and leg. The gearbox ensures that the leg rotates slower than the motor shaft and the torque delivered by the motor to the leg is increased. Again, the parameter values are taken from documentation provided by the manufacturer and are presented in Table 4-4. The gearboxes used are: Maxon Planetary Gearhead GP 32 A Ø32 mm, $0.75 - 4.5$ Nm, order number 166159. The parameter values are given in Table 4-4.



(a) Top view of the model of the robot



(b) Side view of the model of the robot

Figure 4-4: Dimensions of the hexapod model. This figure shows the dimensions of the hexapedal robot as used in the Matlab/Simulink model. The values of the parameters can be found in Table 4-2.

Table 4-3: DC motor parameters

Parameter	Symbol	Value
Armature resistance	R	0.611 Ω
Armature inductance	L	$1.19.10^{-4}$ H
Speed constant	K_e	$2.59.10^{-2}$ Vs/rad
Torque constant	K_t	$2.59.10^{-2}$ Nm/A
Mass inertia	J_m	$3.33.10^{-6}$ kgm^2
Nominal voltage	U_{nom}	24 V
Maximum power consumption	P_{max}	60 W

Table 4-4: Gearbox parameters

Parameter	Symbol	Value
Reduction	n	18 : 1
Mass inertia	J_g	8.10^{-8} kgm^2

Leg-ground contact

In literature many different parameter values of the ground contact have been found and there seems to be no real consensus on how to choose these values. For instance, Taga (1995) modelled an eight-link bipedal robot with the parameters of ground contact equal in both x- and z-direction; where $K_{p,g,x} = K_{p,g,z} = 30.000\ N/m$ and $K_{d,g,x} = K_{d,g,z} = 1000\ Ns/m$. Lee et al. (1998) used values that are 2-5 times higher to model the ground contact of a quadruped robot in 3D, found using the pseudo-inverse solution of the model with closed loop kinematic chains (Kumar and Waldron, 1988). Furthermore, it has been tried to analytically determine the spring and damper constants of the ground, based on studies on soil mechanics, by Silva et al. (2005). They assumed different values for the parameters in x- and z-direction. This method found, for instance for concrete values of over $1.10^9\ N/m$ for the stiffness parameters and values over $1.10^5\ Ns/m$ for the damping parameters. However, these values depend on the dimensions of the robot itself and are therefore not transferable to the model developed in this thesis.

It is therefore necessary to define a method with which the values can be determined. This determination will be done based on the properties of mass-spring-damper systems. The requirements for the parameters are as follows: if the robot is placed on the ground, with only a single leg touching the ground, the displacement of the body should be less than 0.5% and the damping ratio $\zeta > 0.9$. The minimal value for the stiffness can be calculated with

$$K_{p,g,z} = \frac{m_{\text{total}} g}{\Delta z} = \frac{m_{\text{total}} g}{l_{\text{leg}} \times 0.005} = \frac{10.6\ kg \times 9.81\ m/s^2}{0.15\ m \times 0.005} = 138648\ N/m \qquad (4\text{-}42)$$

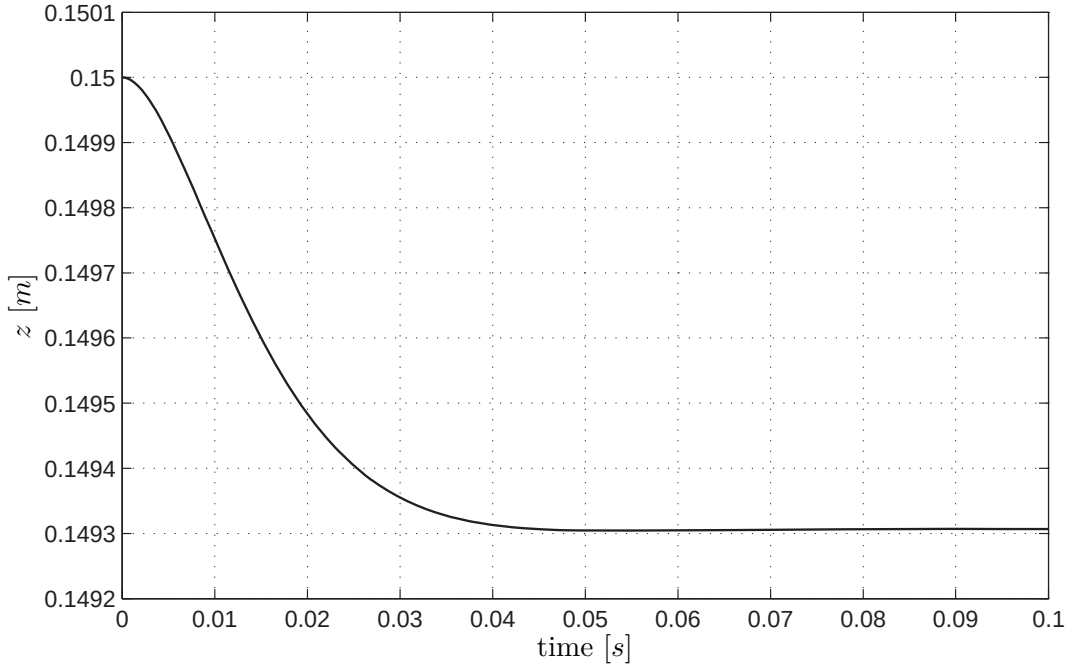


Figure 4-5: The z-position of the CoM of the robot when released from $0.15\ m$, with a single leg touching the ground. This figure is used to determine the optimal parameters for $K_{p,g,z}$ and $K_{d,g,z}$, where the requirements are that the displacement of the body in equilibrium is less than 0.5% and the damping ratio $\zeta > 0.9$. The values of the parameters are given in Table 4-5.

where m_{total} is the total mass of the robot, g is the acceleration due to gravity and l_{leg} is the length of a leg, and thus the height of the robot when placed on the ground. The damping constant can then be calculated using

$$K_{d,g,z} = 2m_{\text{total}}\zeta\sqrt{\frac{K_{p,g,z}}{m_{\text{total}}}} = 2 \times 10.6\ kg \times 0.9 \times \sqrt{\frac{138648\ N/m}{10.6\ kg}} = 2182\ Ns/m \quad (4\text{-}43)$$

where ζ is the damping ratio.

The reason the values are determined with only a single leg touching the ground is that this gives the most extreme situation of the leg-ground contact. In practice more legs will touch the ground at the same time, but the requirements are still satisfied in that case. The value of the stiffness in x-direction is chosen to be equal to the stiffness in z-direction. However, the damping constant is lowered; using the same value introduced unwanted/unrealistic behaviour into the simulation during tests (under certain settings the robot would bounce heavily on the ground.).

The actual values used in the simulation are given in Table 4-5 and the displacement of the body is given in Figure 4-5.

Table 4-5: Parameters of the ground contact forces

Parameter	Symbol	Value
Stiffness x-direction	$K_{p,g,x}$	15.10^4 N/m
Damping x-direction	$K_{d,g,x}$	100 Ns/m
Stiffness z-direction	$K_{p,g,z}$	15.10^4 N/m
Damping z-direction	$K_{d,g,z}$	2200 Ns/m

Controlling the leg positions

The PD-controllers used as reference trajectory trackers output a voltage for the DC motors, which in their turn apply torques to the legs (Section 2-1-2). The PD-controllers thus do not directly generate a torque themselves. The equations of the output of the controller are given by

$$U_i = -K_d\dot{\phi}_i - K_p \sin(\phi_i - \phi_{\text{ref},i}), \qquad i = 1, \ldots, 6 \tag{4-44}$$

where U_i is the voltage applied to the DC motor driving leg i, K_d is the derivative term of the controller, K_p is the proportional term of the controller, ϕ_i is the actual position of leg i, $\phi_{\text{ref},i}$ is the desired position of the leg and $\dot{\phi}_i$ is the angular velocity of leg i. The reason that the sin term is included in this equation is that the variables ϕ_i and $\phi_{\text{ref},i}$ are continuous on $\mathcal{S}^1$, but are projected onto an interval $[-\pi, \pi] \in \mathbb{R}$. If the sin term is not included the leg can be forced to rotate in the wrong direction and in the worst case destabilize the robot. For instance, if $\phi_i = 0.95\pi$ rad and $\phi_{\text{ref},i} = -0.95\pi$ rad the actual difference is only -0.1π rad instead of 1.9π rad.

The test case used to tune the parameters is one in which legs $\{1, 4, 5\}$ and legs $\{2, 3, 6\}$ are kept π rad out of phase and have to follow a ramp signal with a slope of π rad/s, i.e. a leg completes a full rotation in 2 s. This is comparable to the Max-Plus gait generation setting in which the lift-off and touchdown angles differ π rad from each other and the double stance time is zero. However, using a constant velocity input for the reference trajectory instead of Max-Plus offers more freedom during tests. By keeping the legs π rad out of phase the situation is created in which the body has to be lifted from the ground to its top position after which it hits the ground again. Combining this with the ramp input causes the legs to hit the ground very hard; harder than likely to be encountered during the actual experiments.

When tuning the parameters not only how well the legs can follow the reference trajectory have to be kept in mind, but also the voltage output of the PD-controller, as the DC motors are limited to 24 V (see Table 4-3).

The requirement for the controller is that the reference trajectory must be followed at all times with an error of less than 0.1 rad. The reason that this value is chosen quite large is to not saturate the DC motors. The values chosen for K_d and K_p are presented in Table 4-6. In Figure 4-6 the reference trajectory and position of one of the legs are

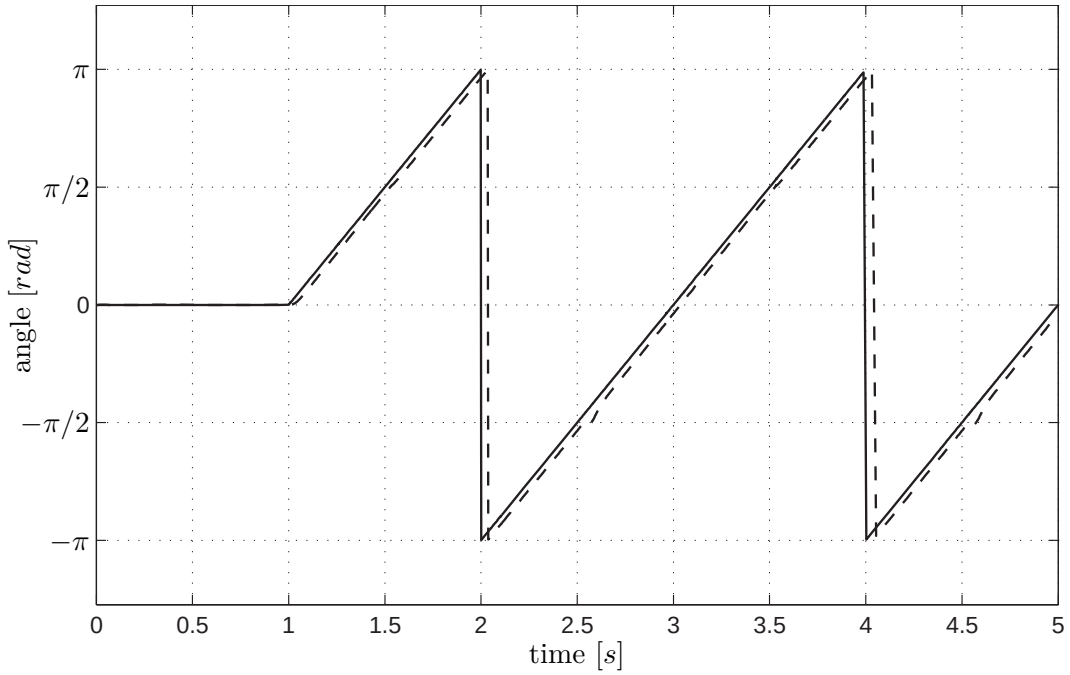


Figure 4-6: A comparison of a reference trajectory (solid line) and the actual position of a leg (dashed line). The figure is generated using a tripod gait with leg sets $\{1, 4, 5\}$ and $\{2, 3, 6\}$, with a difference of π rad between the two sets and a ramp signal with a slope of π rad/s for all legs. The reference trajectory shown is of leg 1. As can be seen the PD-controller is able to let the leg follow the reference trajectory very well, with an error of less than 0.1 rad. The larger errors at approximately 2.5 s and 4.5 s are caused by the leg hitting the ground and are acceptable.

shown and in Figure 4-7 the corresponding voltage output is shown. As can be seen the PD-controller is generally able to let the leg follow the reference trajectory within the error bounds and the voltage is within the limit. However, this is not the case at the moment of ground impact. The position error grows to 0.2 rad and the voltage rises well above 24 V. This behaviour cannot be avoided and must be taken for granted. A noteworthy result from the tuning process is that if the damping is too low the robot is not able to stand; it will collapse under his own weight.

Table 4-6: PD-controller parameters

Parameter	Symbol	Value
Derivative term	K_d	10 Us/rad
Proportional term	K_p	300 U/rad

4-4 Conclusions

In this chapter the structure of the optimization process was defined, which includes the states, actions and rewards used. In order to reduce the number of states present in

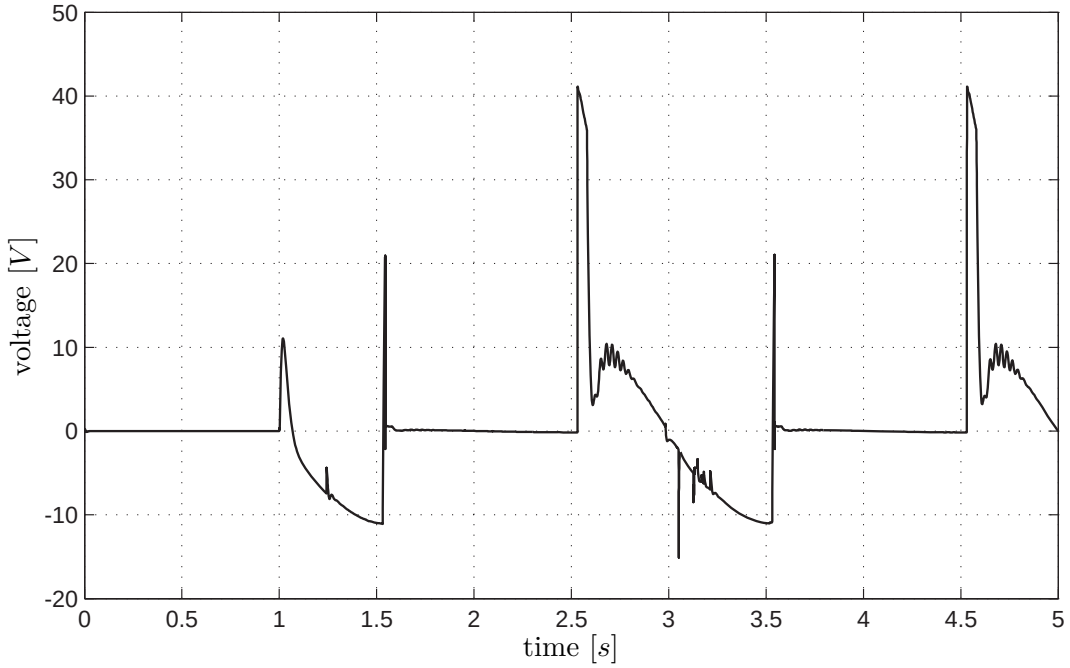


Figure 4-7: The output of the reference trajectory controller. This figure shows the voltage output of a PD-controller. The figure is generated using a tripod gait with leg sets $\{1, 4, 5\}$ and $\{2, 3, 6\}$, with a difference of π rad between the two sets and a ramp signal with a slope of π rad/s for all legs. The voltage shown is of leg 1. As can be seen the voltage crosses the 24 V level only when the leg is hitting the ground, which is something that cannot be prevented, but in general it stays well below this level.

the state vector the concept of the virtual leg was introduced. The position and angular velocity of this virtual leg are defined as the average of the position and angular velocity of all legs present in the system, respectively. This effectively reduces the problem of a six-legged robot to that of a robot with a single leg.

The actions applied to the system are not the voltages applied to the DC motors, as might seem to be the obvious choice, but the lift-off and touchdown angles of the legs used in the reference trajectory generator, which offers a more direct and intuitive approach. The actions follow from a parametrized stochastic policy, which was obtained by adding a value from a normal distribution to a deterministic policy. The deterministic policy was build up from polynomial functions used for the non-periodic states and Fourier functions used for the periodic states present in the state vector. The latter is a novel method of dealing with periodic states in the policy; no records have been found in literature which use this kind of dependency of the actions on the states.

To determine how good a certain policy parameter set is two different types of reward functions were introduced: based on the weighted average of the velocity and power consumption, and based on the specific resistance. The latter is a dimensionless quantity often used in literature to compare different types of moving objects to each other. This concludes the structure of the optimization problem as it will be used in the experiments described in Chapter 5.

Chapter 5

Experimental validation

In this chapter the theory as presented in Chapter 2 and Chapter 3, together with the structure of the learning problem as defined in Chapter 4, is applied in simulation in order to optimize gait properties of a hexapod robot. Before the experiments themselves are discussed, it is necessary to define settings for the learning problem that will be used throughout all experiments. This is done in Section 5-1, which can be divided in a part dealing with the Switching Max-Plus-linear model and a part dealing with the episodic Natural Actor-Critic (eNAC) method. The first experiment performed, Section 5-2, is maximizing the velocity of the robot in forward direction. During this experiment the robot is walking on a completely flat terrain and no external disturbances are present. The second experiment, Section 5-3, also deals with the maximization of the forward velocity. However, this time the robot is walking over a terrain of which the height varies. Finally, in Section 5-4 the overall conclusions from the experiments are listed.

5-1 Implementation considerations

In this section an overview is given of the settings that are used throughout all experiments. The settings can be divided in two parts; one dealing with the Switching Max-Plus-linear model and one dealing with the eNAC method.

Switching Max-Plus-linear model

An overview of the general Switching Max-Plus-linear model settings used is given in Table 5-1. The gait for all experiments is the so-called tripod gait (Section 2-2-2) in which two leg sets, each containing three legs, are defined. The flight time τ_f is chosen to be 0.6 s and the double stance time τ_Δ is chosen to be 0.2 s, resulting in a cycle time of the legs of 1.6 s.

Table 5-1: General settings Switching Max-Plus-linear model

Parameter	Symbol	Value
Gait		$\{\{1,4,5\},\{2,3,6\}\}$
Flight time	τ_f	0.6 s
Double stance time	τ_Δ	0.2 s

episodic Natural Actor-Critic

An overview of the general eNAC settings used is given in Table 5-2. The policy is equal for both parameters in terms of the function (not in terms of the parameters themselves). The order of the polynomial functions and the Fourier functions is chosen to be 2 and a constant parameter is used; no bilinear part is defined, which results in a total of 44 parameters. Assuming some unknown (high order) relation between states and actions, using a bilinear part and/or higher order functions could yield a better result. However, this also increases the complexity of the problem quadratically (see Section 3-2-1) and a trade-off must thus be made.

Table 5-2: General settings episodic Natural Actor-Critic

Parameter	Symbol	Value
Constant in policy		yes
Bilinear part in policy		no
Order polynomial functions in policy		2
Order Fourier functions in policy		2
Threshold parameter update	ε_w	10 °
Horizon parameter update	τ_w	4
Standard deviation start state	σ_{start}	0.04
Standard deviation policy	$\sigma_{\phi_l}, \sigma_{\phi_t}$	0.02

There is very little intuition on how to initialize the parameters of the policy correctly, i.e. on how the lift-off and touchdown angles depend on the different states of the robot. A possible relation that exists is between the pitch of the body and the lift-off and touchdown angles. The intuition is that if the body of the robot is rotated to a large extend, the legs must rotate accordingly in order to "catch" the robot and make sure the body does not hit the ground[1]. However, to be safe and to not introduce incorrect relations into the initial policy it is chosen only to initialize the constant parameters with 0.4 and -0.4 for lift-off and touchdown, respectively.

[1] This is something that comes from the observation of the author on how humans react when tripping while walking and balance cannot be restored, causing a person to fall over. The persons first reaction is to extend his or her arms in the direction of the ground to reduce the impact of the fall and protect their body.

The exploration of the policy (Section 4-2-1) is set equal for both the lift-off and touchdown angles at the start of the optimization. As these parameters are included in the parameter vector of the policy, these values will vary during the optimization. A value of 0.02 for the standard deviation is chosen, which equals a maximum change of 10% of the initial values of the lift-off and touchdown angles in 95.4% of the actions applied[2].

A last important aspect of the policy follows from the way in which the reference trajectories of the legs are generated (Section 2-2-2), which requires that $\phi_t < \phi_l$. However, this is something that cannot be checked before performing an episode, due to the large, continuous state space. Therefore, it is chosen to discard the results of an optimization in which this requirement is violated, when exploration is turned off[3].

In order to reduce the optimization time it is chosen not to define a threshold and horizon for the stopping criterion of the optimization, in contrast to suggested in Section 4-2-2, but to fix the number of parameter updates to 20. Although the optimization might not be fully completed at the maximum number of parameter updates, the results give an indication whether or not an improvement is found and the method works. The stopping criteria for a parameter update are defined as 10° for the threshold and 4 episodes for the horizon. The horizon of 4 is chosen as from tests it is found that sometimes in the beginning of the update process 2 or 3 subsequent vectors can point roughly in the same direction. However, when running the update process for a larger number of episodes it is found that for converge more episodes are needed. The relatively large threshold is chosen as from tests it is found that the angle between subsequent parameter updates grows smaller with increasing number of episodes, but due to the stochastic nature of the process some outliers are possible. Combining this with the horizon, the threshold cannot be made smaller in order for the process to converge within a reasonable number of episodes.

Table 5-3: Start state values

Parameter	Symbol	Value
z-position	z	0.15 m
Angle body	β	0 rad
Angle virtual leg	$\bar{\phi}$	0 rad
Velocity in x-direction	$\dot{x}$	0 m/s
Velocity in z-direction	$\dot{z}$	0 m/s
Angular velocity body	$\dot{\beta}$	0 rad/s
Angular velocity virtual leg	$\dot{\bar{\phi}}$	0 rad/s
Phase Max-Plus gait generation	φ	0 rad

[2] This follows from the normal distribution theory where 95.4% of the values will be within $\mu \pm 2\sigma = 0.4\ rad \pm 0.04$.

[3] It is chosen to check this with exploration turned off as, due to the stochastic nature of action selection, the requirement can be violated by accident during an episode. However, the remainder of the episodes might satisfy the requirement and it is therefore not necessary to discard the entire optimization run.

As given in Algorithm 3-3 it is necessary to draw the start state from a probability distribution. A normal distribution, with standard deviation σ_{start} and as mean the starting value of each state is used. However, due to the nature of the system it is not possible to vary all states at the start. This is the case for the z-position and the angle β of the body. When varying these values the springs at the ground contact points can become "loaded" with energy causing the robot to "jump up" or in the worst case even to be launched, which is behaviour that must be avoided. This holds to a far less extent for the velocity $\dot{z}$ and the angular velocities $\dot{\beta}$ and $\dot{\bar{\phi}}$ due to smaller constants[4], and these can therefore be drawn from a probability distribution. Furthermore, the phase φ cannot be varied as it will always start at 0. For simplicity, the same standard deviation is used for all remaining states. The start state values are given in Table 5-3.

Finally, for all experiments the average reward setting is used. The discounted return does not seem to be appropriate to use, as it cannot be said that actions performed at the beginning of each episode are more important than actions performed at the end of the episode, due to the cyclic nature of the process. The reward functions as defined in Section 4-1-3 are all given in continuous time. However, the eNAC method will only collect data at certain time intervals, i.e. in discrete time. It is therefore necessary to transform the rewards received in continuous time to discrete rewards in order to be used by the optimization. This is done by integrating the continuous rewards over the discrete time step size and dividing this by the step size. This can be expressed in the following equation

$$r_k = \frac{\int_t^{t+\Delta t} r_t \Delta t}{\Delta t} \tag{5-1}$$

where k denotes the discrete time step and t denotes continuous time.

5-2 Experiment 1 - maximizing the velocity, flat terrain

The first experiment that is performed is maximizing the velocity of the robot in x-direction. In Section 5-1 the general settings of the experiments performed are discussed, but there are some specific settings for this experiment, which are defined in Section 5-2-1. After that an overview of the results is given in Section 5-2-2. This includes a comparison of the states and actions before optimization, as well as a comparison of several properties such as power consumption and specific resistance. Furthermore, the parameter vectors obtained after optimization are analysed and the influence of the states on the actions is deduced. Finally, in Section 5-2-3 the results are discussed and conclusions are drawn.

[4] A compression of the springs in z-direction of e.g. 0.02 m will result in a far larger force than when the body is moving down with 0.02 m/s. Even a compression of only 0.001 m, assuming no rotation of the body and the legs all pointing downwards, will result in an external force of more than 800 N (Table 4-5), which is about a factor 8 times larger than the force in opposite direction due to gravity.

5-2-1 Experiment specific settings

An overview of experiment specific settings is given in Table 5-4. The terrain on which the robot is walking is completely flat. To optimize the velocity the weighted average reward function needs to be used, with the weight $\alpha = 1$ (Section 4-1-3).

An important setting of the eNAC method that has not been defined yet is the sampling rate, i.e. how often is data collected from, and are actions applied to the system. Two questions must be answered to establish a sampling rate: how do the states vary during a run and how do the actions influence the system, to begin with the former. In Figure 5-2 an overview is given of all states before optimization, using the initial policy as described in Section 5-1, but exploration turned off[5]. What follows directly from these figures is that the movement of the robot is very stable, e.g. there are no large variations in pitch angle β of the body and the velocity $\dot{x}$ of the robot is fairly constant except for some large spikes at lift-off and touchdown moments.

Table 5-4: Settings Experiment 1

Parameter	Symbol	Value
Weighted average weight	α	1
Sampling rate		1.25 Hz
Simulation time		16 s
Learning rate actor	α_a	0.025
Number of runs		30

The second question to answer is how the actions influence the system. Due to the fact that the ground is flat, the robot is walking in a very stable and constant manner as described before; the legs will always touch the ground at the desired moment, assuming accurate reference tracking. This makes it unnecessary to constantly change the actions, i.e. to sample the system at high frequency. The sampling frequency chosen is

$$f = \frac{2}{\lambda} = 1.25 \qquad [Hz] \tag{5-2}$$

where λ is the cycle time of a leg. This gives that the system is sampled twice per cycle time, thus changing the actions twice. In order to collect enough data for the eNAC method to estimate a gradient, the episode length is set to 10 times the cycle time of a leg, i.e. the robot will walk for 20 steps within an episode collecting 20 data samples[6].

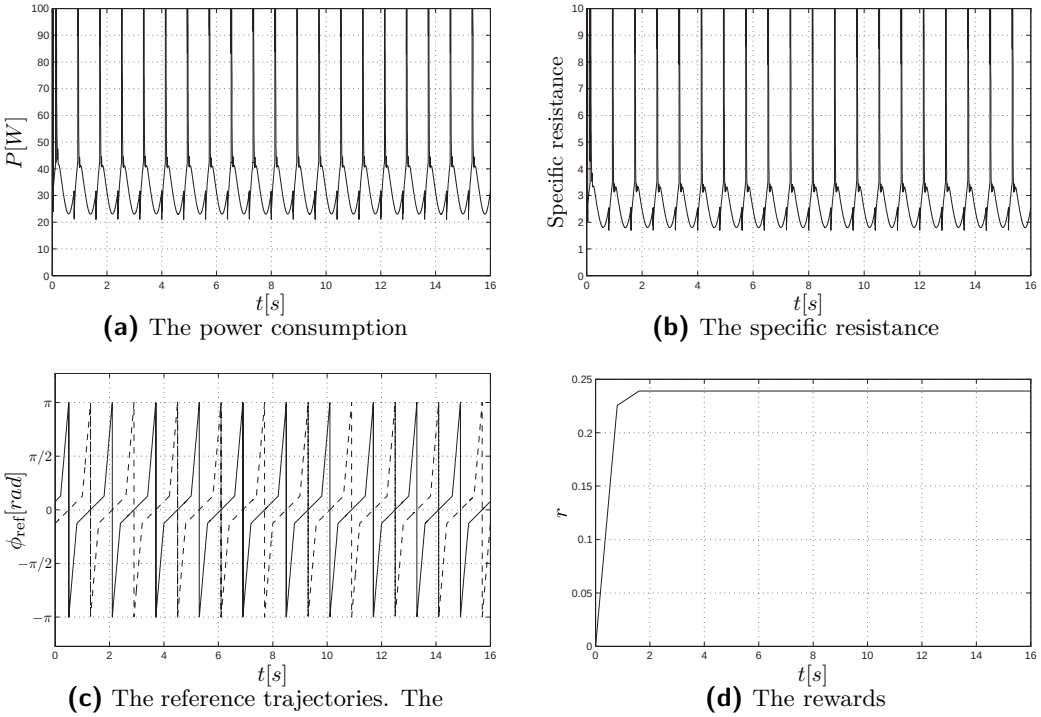
[5] In all figures an episode length of 10 cycle times, which equals 16 s, is considered. The decision for this is motivated later on in this section.

[6] In order to derive the number of samples needed use is made of Peters and Schaal (2008b). In the problem described, a baseball hitting a ball, there are a total of 14 continuous states, 7 continuous actions and 70 policy parameters present. A sample rate of 60 Hz is used and the estimated length of an episode is a second, resulting in roughly 60 samples collected during an episode. Due to the fact that the problem considered in this thesis contains less states, actions and parameters, a lower number of samples is used.

Furthermore, it is necessary to define the learning rate α_a of the actor (Algorithm 3-3), which is chosen to be 0.025. Finally, the number of runs performed within the experiment is 30 in order to obtain statistically meaningful data[7].

The actions of the system are, before optimization, constants: 0.4 rad and -0.4 rad for lift-off and touchdown, respectively, and are therefore not shown in a figure. To complete the analysis of the system before optimization, plots of the power consumption, the specific resistance, the reference trajectories of the legs and the rewards are shown in Figure 5-1.

The initial average return, using no exploration in the policy, is 0.2285, which corresponds to an initial average velocity of 0.11 m/s. The specific resistance of the robot is given by 3.63, comparable to "Big Muskie" shown in Figure 1-2[8].



(a) The power consumption

(b) The specific resistance

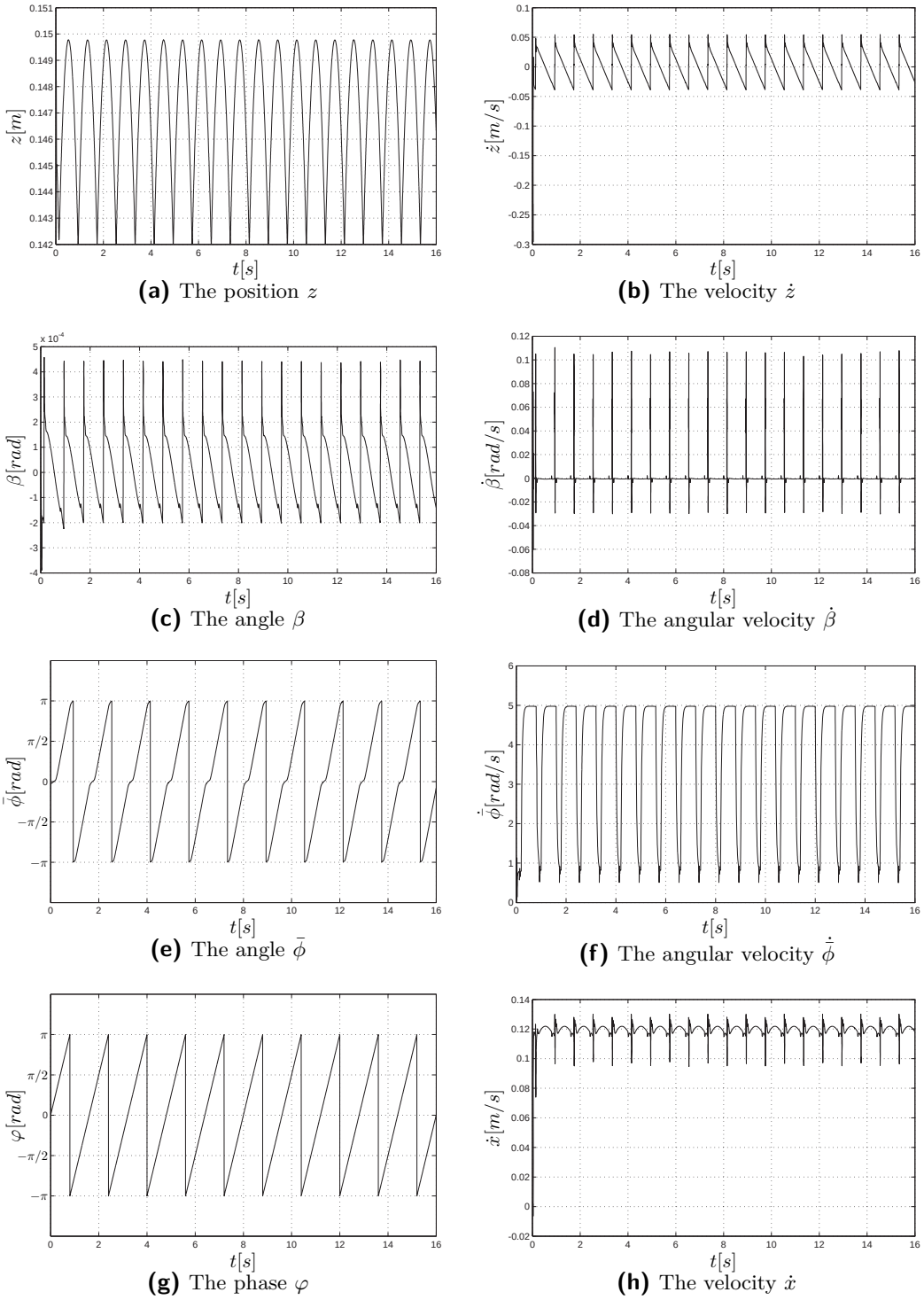
(c) The reference trajectories. The solid line denotes the reference trajectory for legs $\{1, 4, 5\}$, the dashed line denotes the reference trajectory for legs $\{2, 3, 6\}$.

(d) The rewards

Figure 5-1: Experiment 1 - maximizing the velocity, flat terrain. Initial properties. These figures show some of the properties before optimization. The plots of the power consumption and specific resistance are zoomed in and do not show the entire range. The rewards shown are the sampled rewards, with sampling frequency 1.25 Hz.

[7] In this thesis a series of parameter updates is referred to as a run. We thereby have the following hierarchy, from low to high: episode → parameter update → run → experiment.

[8] Due to the fact the specific resistance becomes very high at some moments in time, the values are obtained by integrating the specific resistance over time in a similar fashion in which the discrete rewards are obtained (Eq. (5-1)). However, the first cycle is not taken into account, as the system needs to settle first.



(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\bar{\phi}$ **(f)** The angular velocity $\dot{\bar{\phi}}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-2: Experiment 1 - maximizing the velocity, flat terrain. Initial states. These figures show the states of the robot before optimization, using the initial policy as described in Section 5-1, but exploration turned off. The movement of the robot is very smooth, e.g. there are no large variations in pitch angle β of the body and the velocity $\dot{x}$ of the robot is fairly constant. Furthermore, it can be seen that for an undisturbed system, with constant actions applied, the virtual leg approximately equals the phase of the Max-Plus gait generation.

5-2-2 Results

In this section the results obtained from the simulations are presented. As follows from Section 5-2-1 a total of 30 runs were performed within this experiment. However, as explained in Section 5-1 it must first be checked if the requirement $\phi_t < \phi_l$ is violated, and if so, the results of that particular run must be discarded. It is found that out of 30 runs only one violated the constraint, leaving 29 runs. When checking the data it was found that one run resulted in a very large return of 0.7249, which is an increase of over 200% from the initial return. However, due to a slight error in the determination of the body height there was no negative reward given to the agent although the body touched the ground at some occasions[9]. Therefore, this result is not taken into account when determining the statistical properties of the results. However, this run is included in parts of the analysis of the results as it has some surprising implications[10].

The mean and standard deviation of the returns after each parameter update are shown in Figure 5-3. In order to visualize Figure 5-3 the returns of all runs are shown in Figure 5-4. From Figure 5-3 it can be seen that the mean of the return increases from 0.2292, before a parameter update is performed[11], to 0.2682 after the last parameter update, which is an increase of 17%. In terms of average velocity this means an increase from 0.11 m/s to 0.13 m/s.

All runs consisted of 20 parameter updates, and the average number of episodes in a run is found to be 946. In each episode 10 complete leg cycles are considered, giving a total simulated time of 15136 s or approximately 4 hours and 12 minutes per run. However, from Figure 5-4 it follows that most runs approximately converge within 6 parameter updates, reducing the average number of episodes to 270, which equals 4320 s or 72 minutes per run. Furthermore, the robot walked for approximately 2 meters per episode, giving a total walking distance of roughly 540 meters. The specific resistance before optimization was given by 3.63 and after optimization it is found to be 3.80.

[9] At most a corner of the body was at height $z = -0.021$ m, which is less then half the height of the body.
[10] It will be clearly stated in the text if this run is included.
[11] The reason that it differs from the value 0.2285 given earlier in this section, is that 0.2292 follows from using a stochastic policy and 0.2285 follows from the deterministic policy.

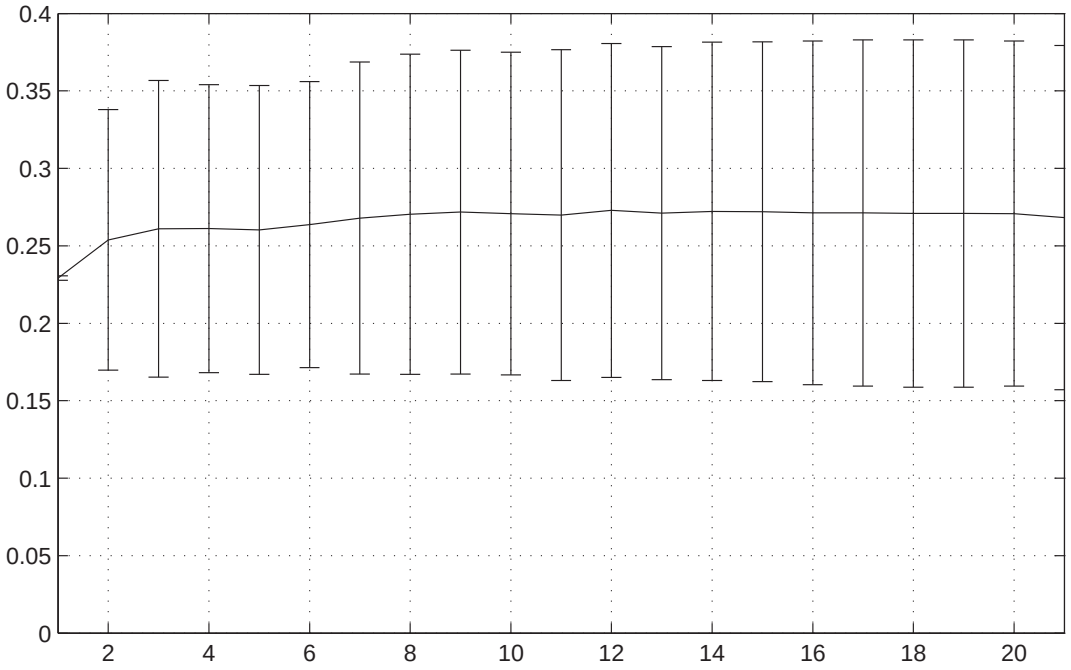


Figure 5-3: Experiment 1 - maximizing the velocity, flat terrain. Mean and standard deviation. This figure shows the mean μ and standard deviation σ of the returns after each parameter update. The mean is given by a solid line and the standard deviation is represented using error bars.

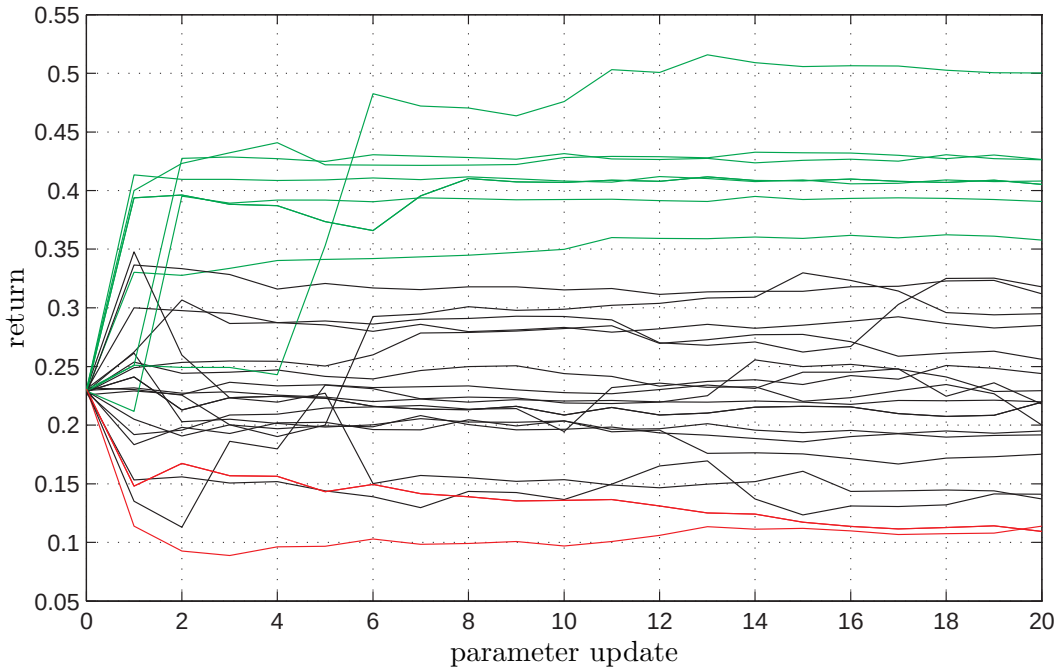


Figure 5-4: Experiment 1 - maximizing the velocity, flat terrain. Returns. This figure shows the (intermediate) average returns of all episodes within a parameter update. Runs from which the last return lies within the $\mu \pm \sigma$ range are shown in black, if the last return lies above $\mu + \sigma$ it is shown in green and if the last return lies below $\mu - \sigma$ it is shown in red.

Comparing the situations before and after optimization

To compare the behaviour of the robot before optimization to the behaviour after optimization, use is made of the run that is closest to the mean as shown in Figure 5-3. The actions are shown in Figure 5-5, in Figure 5-6 the states of the system are compared and in Figure 5-7 properties such as the power consumption, the specific resistance, the reference trajectories of the legs and the rewards are compared.

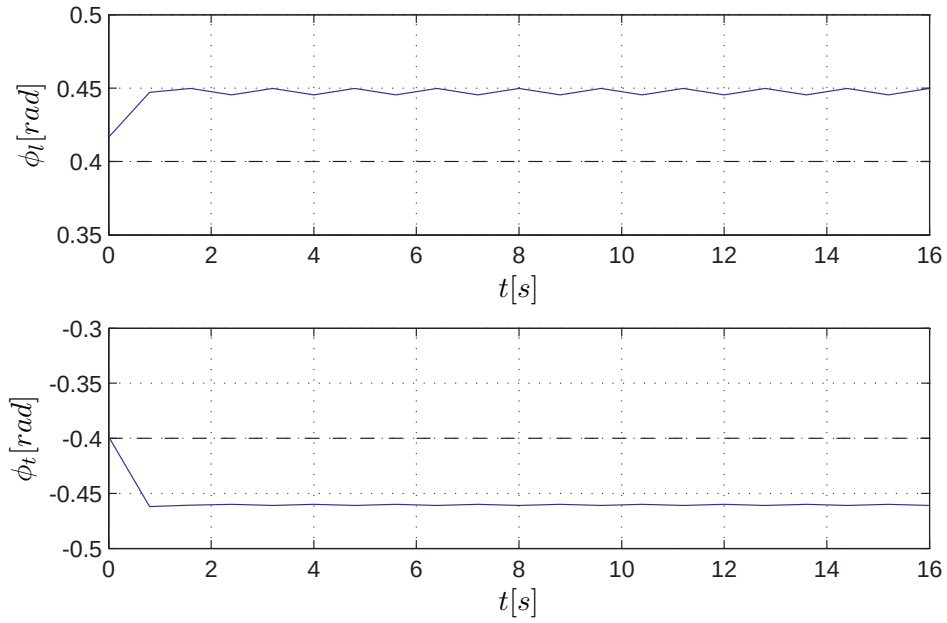
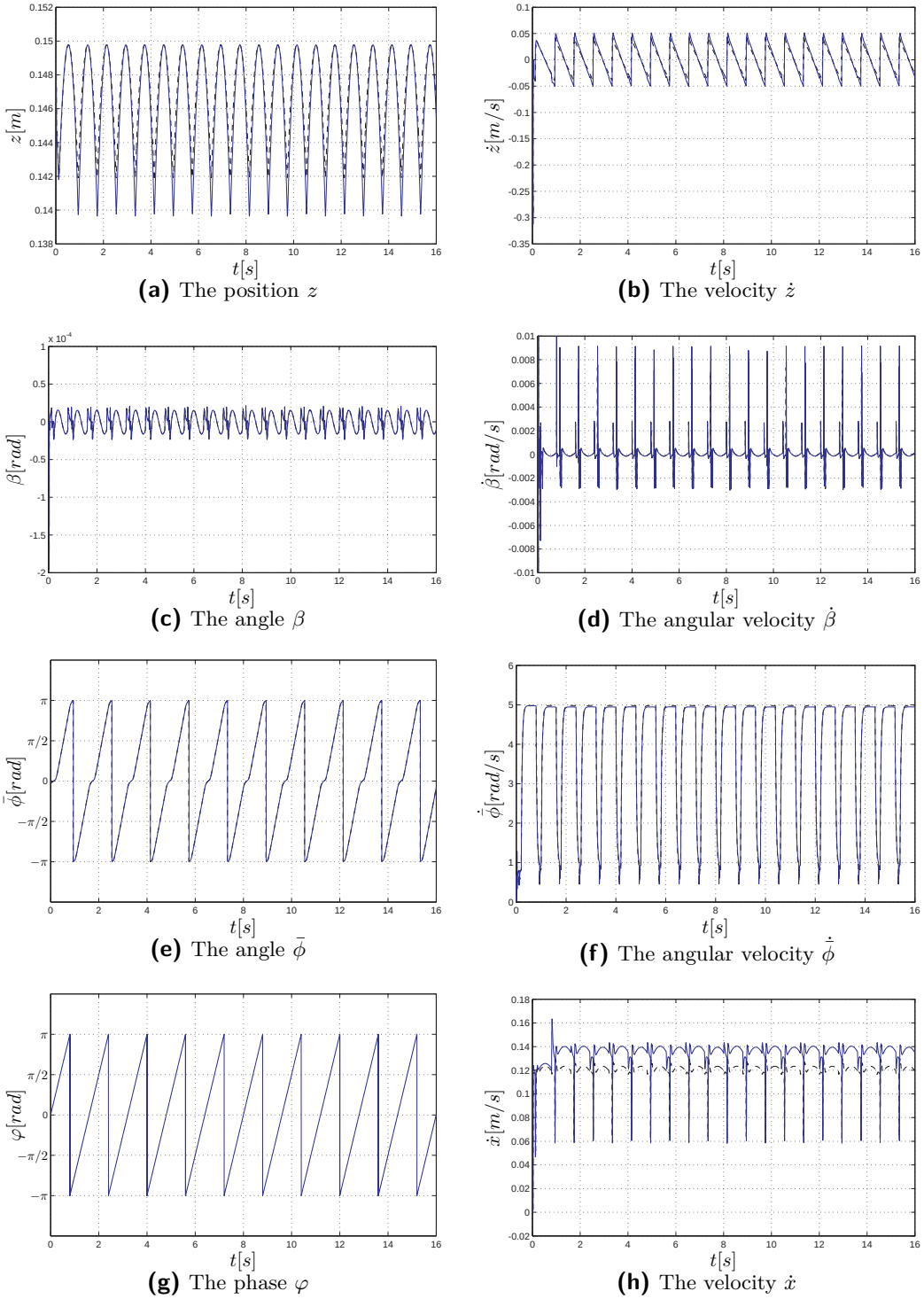


Figure 5-5: Experiment 1 - maximizing the velocity, flat terrain. Comparing the actions before and after optimization. This figure compares the actions before optimization (dashed black lines) to the actions after optimization (solid blue lines). As can be seen the distance between the lift-off and touchdown angles increases.



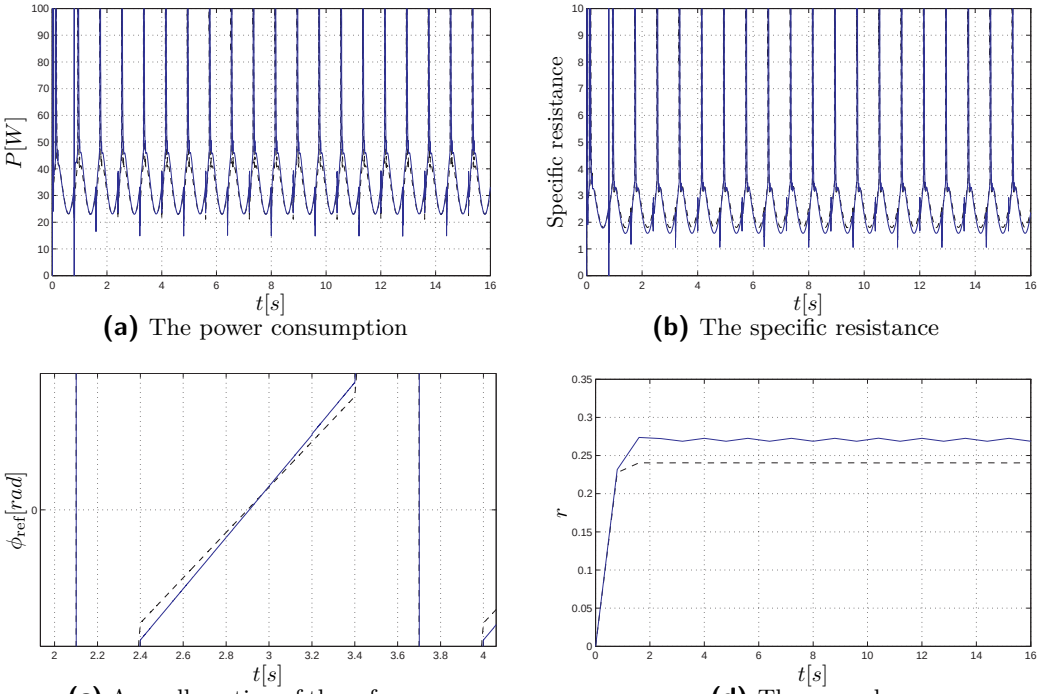
(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\bar{\phi}$ **(f)** The angular velocity $\dot{\bar{\phi}}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-6: Experiment 1 - maximizing the velocity, flat terrain. Comparing the states before and after optimization. These figures show the states of the robot before (dashed black lines) and after optimization (solid blue lines). As can be seen most states stay approximately equal. Only a large difference can be seen in the velocity $\dot{x}$, and smaller differences in $\dot{\bar{\phi}}$, z and $\dot{z}$.



(a) The power consumption

(b) The specific resistance

(c) A small portion of the reference trajectories for legs $\{1, 4, 5\}$

(d) The reward

Figure 5-7: Experiment 1 - maximizing the velocity, flat terrain. Comparing properties before and after optimization. These figures compare some of the properties of the system before (dashed black lines) and after (solid blue lines) optimization. The plots of the power consumption and specific resistance are zoomed in and do not show the entire range. The rewards shown are the sampled rewards, with sampling frequency $1.25\ Hz$. As can be seen the power consumption increased at the peaks, and the specific resistance decreased at the valleys. The change in reference trajectories show that the legs rotate over a larger angle when on the ground after optimization.

Influence of the parameters on the policy

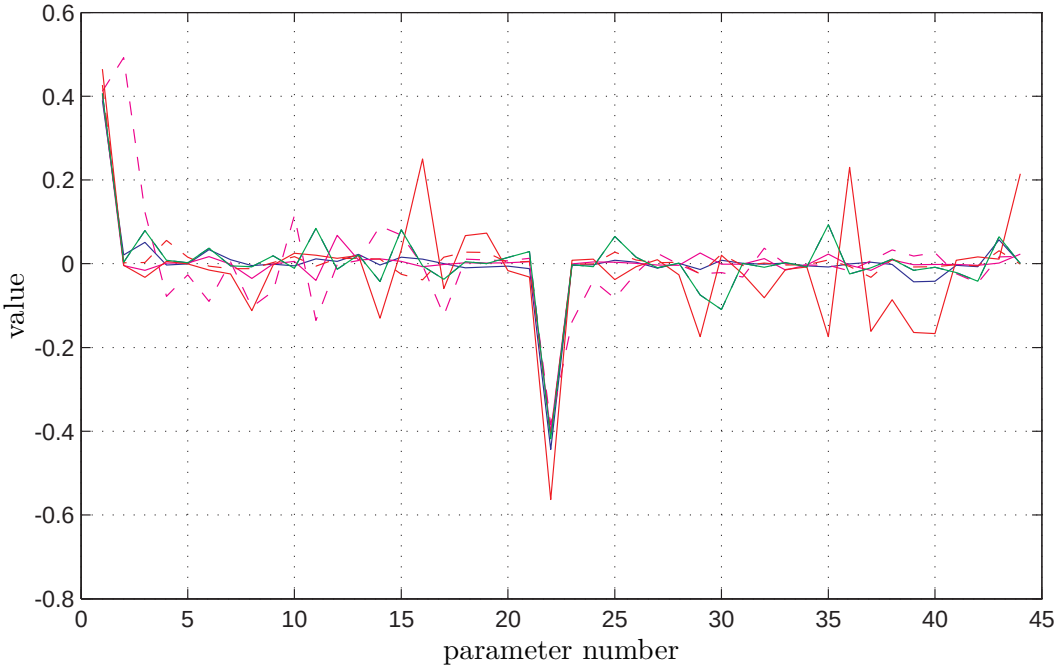
Although the results previously presented hold for the run in which its returns are closest to the mean value of all runs, the most interesting information with respect to the policy parameters can be obtained from the runs that resulted in the highest return. Therefore, in Figure 5-8a a plot is shown of the parameter values after optimization of the 7 runs that resulted in a final return larger than $\mu + \sigma$, which are the green lines shown in Figure 5-3. In Table 5-5 an overview is given of the parameter numbers, as used in Figure 5-8, compared to what they represent or to which state they relate[12]. From this table it can be seen that the first 21 parameters plus parameter 43 belong to action ϕ_l and parameters 22 to 42 and 44 belong to action ϕ_t.

The values for each parameter shown Figure 5-8a vary from parameter vector to parameter vector and do not show a clear relation. Except for parameters 1 and 22, which represent the constants and were initialized with a non-zero value. Therefore, in Figure 5-8b the parameter values corresponding to the run with the highest return are shown. The influence of each state on the actions is given in Figure 5-9. For the range of each state the maximum and minimum values as occurred in Figure 5-6, after optimization, are used.

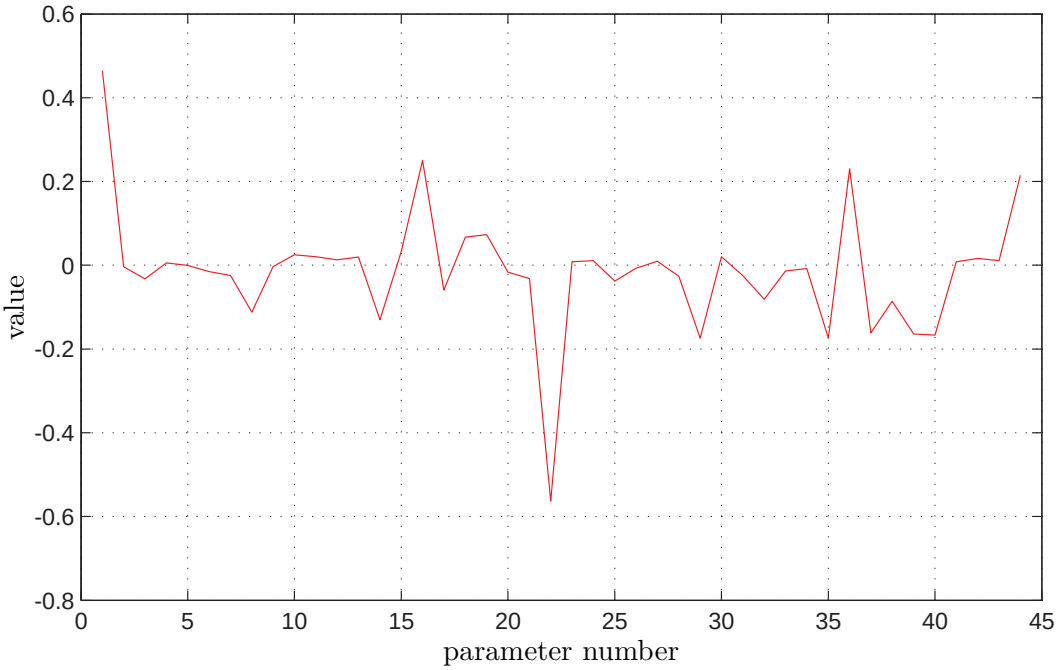
Table 5-5: Relation between policy parameter values and states/representation

	Parameter numbers	
Action / Representation	ϕ_l	ϕ_t
constant	1	22
$\dot{x}, \dot{x}^2$	2,3	23,24
z, z^2	4,5	25,26
$\dot{z}, \dot{z}^2$	6,7	27,28
β, β^2	8,9	29,30
$\dot{\beta}, \dot{\beta}^2$	10,11	31,32
$\dot{\bar{\phi}}, \dot{\bar{\phi}}^2$	12,13	33,34
$\cos(2*\pi*\bar{\phi}/\lambda), \sin(2*\pi*\bar{\phi}/\lambda),$ $\cos(4*\pi*\bar{\phi}/\lambda), \sin(4*\pi*\bar{\phi}/\lambda)$	14,15,16,17	35,36,37,38
$\cos(2*\pi*\varphi/\lambda), \sin(2*\pi*\varphi/\lambda),$ $\cos(4*\pi*\varphi/\lambda), \sin(4*\pi*\varphi/\lambda)$	18,19,20,21	39,40,41,42
$\sigma_{\phi_l}, \sigma_{\phi_t}$	43	44

[12] The order of states is different than to the order of the states in the state vector, due to a different implementation in the simulation.

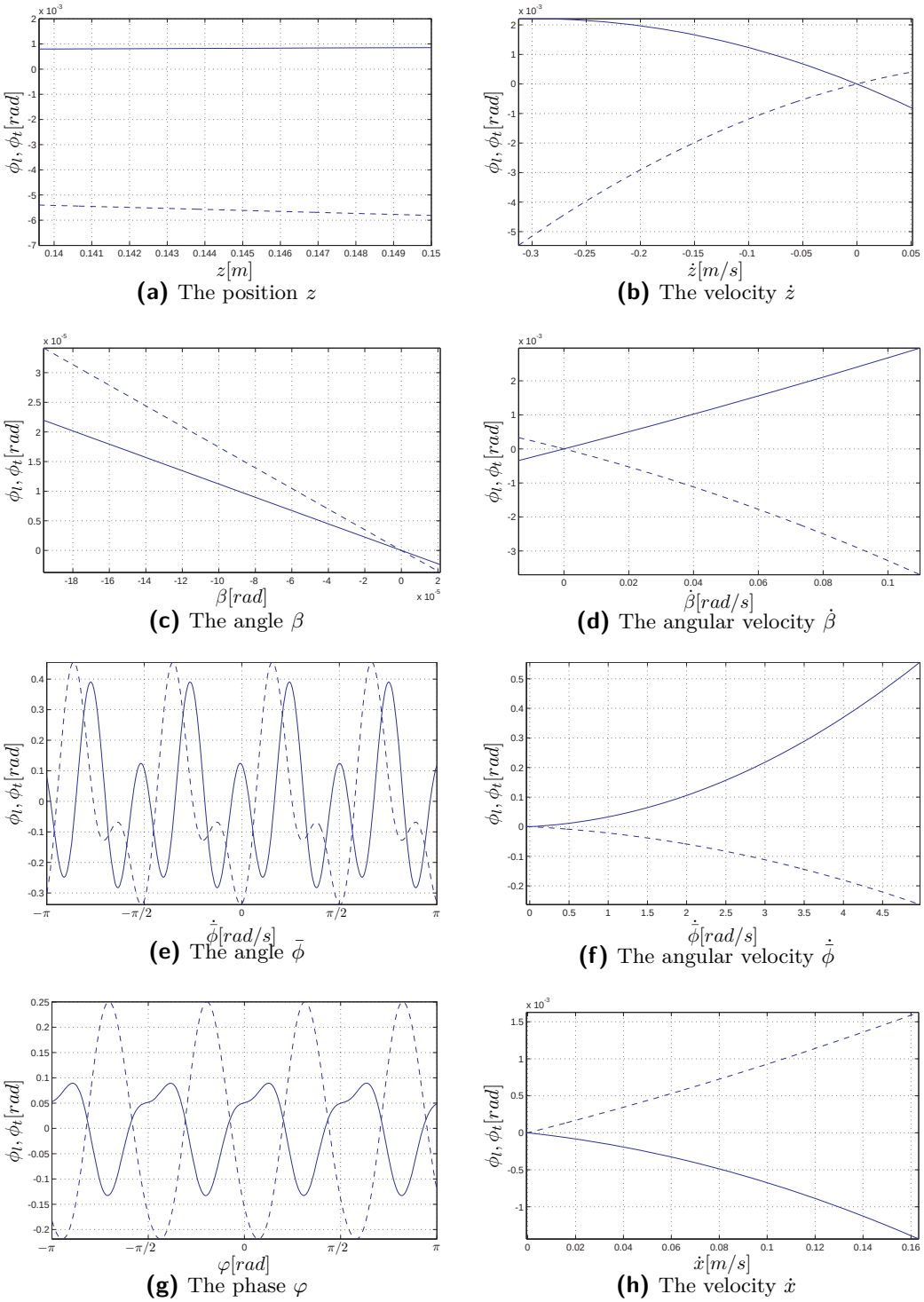


(a) The parameter values corresponding to the 7 runs with the highest return.



(b) The parameter values corresponding to the run with the highest return.

Figure 5-8: Experiment 1 - maximizing the velocity, flat terrain. Parameter values after optimization. These figures show the values of the parameters after optimization of the 7 runs that resulted in a final return larger than $\mu + \sigma$ shown in Figure 5-3, and separately the parameter values belonging to the run with the highest return. To which state/policy part each number corresponds, see Table 5-5.



(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle ϕ **(f)** The angular velocity $\dot{\phi}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-9: Experiment 1 - maximizing the velocity, flat terrain. Influence of states on actions. These figures show the influence of the states on the actions. The solid lines denote the lift-off angle and the dashed lines denote the touchdown angle. For the range of each state the maximum and minimum values as occurred in Figure 5-6, after optimization, are used.

Aerial phase

As mentioned in the beginning of this section there was one run that showed a particularly large increase in return, from 0.2293 before optimization to 0.7249 after optimization. This corresponds to an increase of velocity from 0.11 m/s to 0.35 m/s. The specific resistance is 9.21 for this particular run. In terms of specific resistance this run is placed near the "OSU hexapod" in Figure 1-2, making it a not very efficient walk.

However, at some time instances the body would have been touching the ground, giving a negative reward, but as this was not detected by the simulation the result was removed from previous analysis[13]. The reason this run is particularly interesting is that the robot achieves a completely aerial phase during an episode. The properties of this run are presented in a similar fashion as previous results and are shown in Figure 5-10 to Figure 5-13. The influence of each state on the actions is given in Figure 5-14. For the range of each state the maximum and minimum values as occurred in Figure 5-11, after optimization, are used.

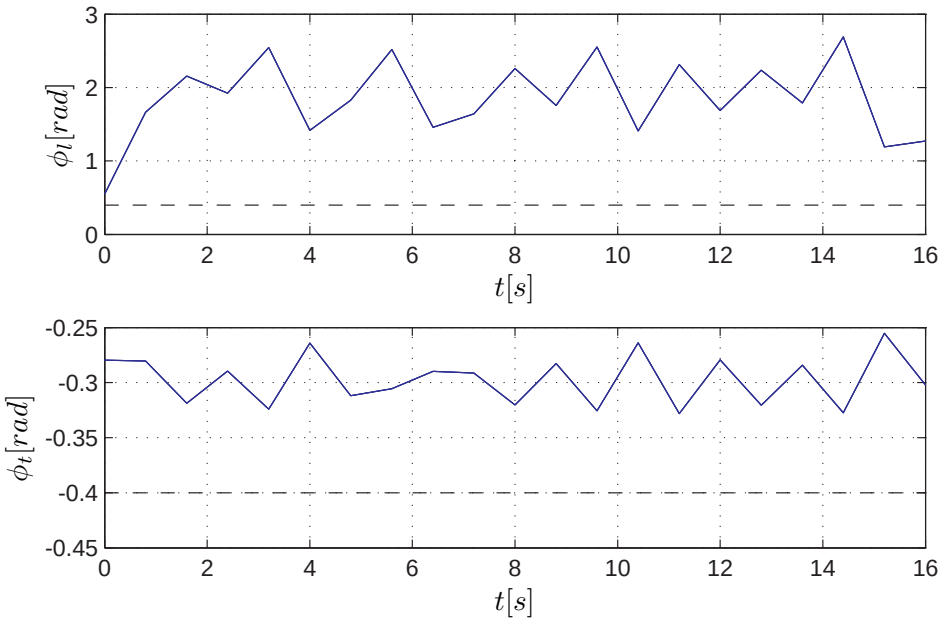
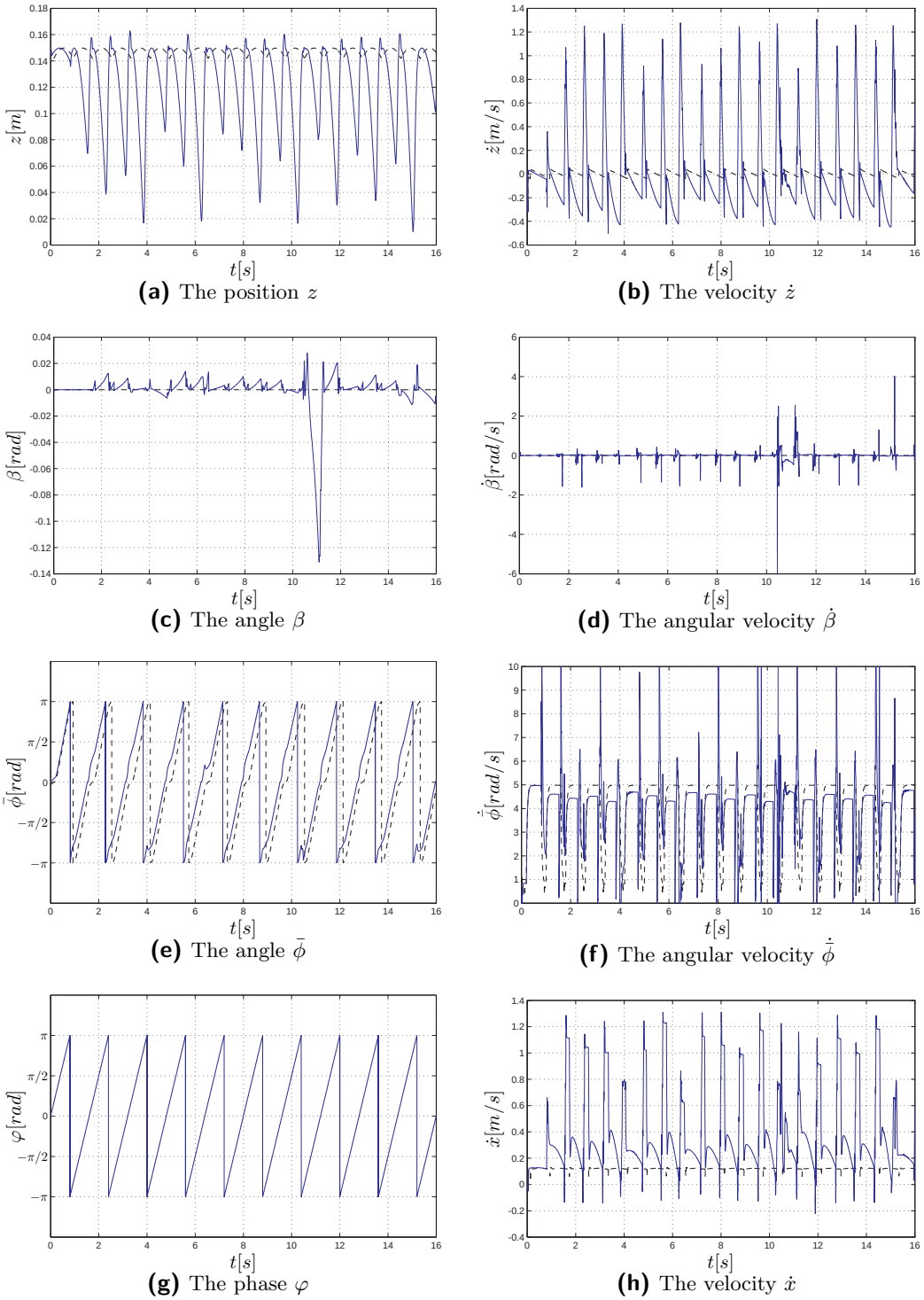


Figure 5-10: Experiment 1 - maximizing the velocity, flat terrain. Actions during run in which an aerial phase is achieved. This figure compares the actions before optimization (dashed black lines) to the actions after optimization (solid blue lines). As can be seen there is a large change in lift-off angle at each sample time.

[13] If the body is replaced by an infinitely thin rigid body with the same mass and moment of inertia, thus not influencing the dynamics of the model, the body would not be touching the ground, making it possible to analyse this run.



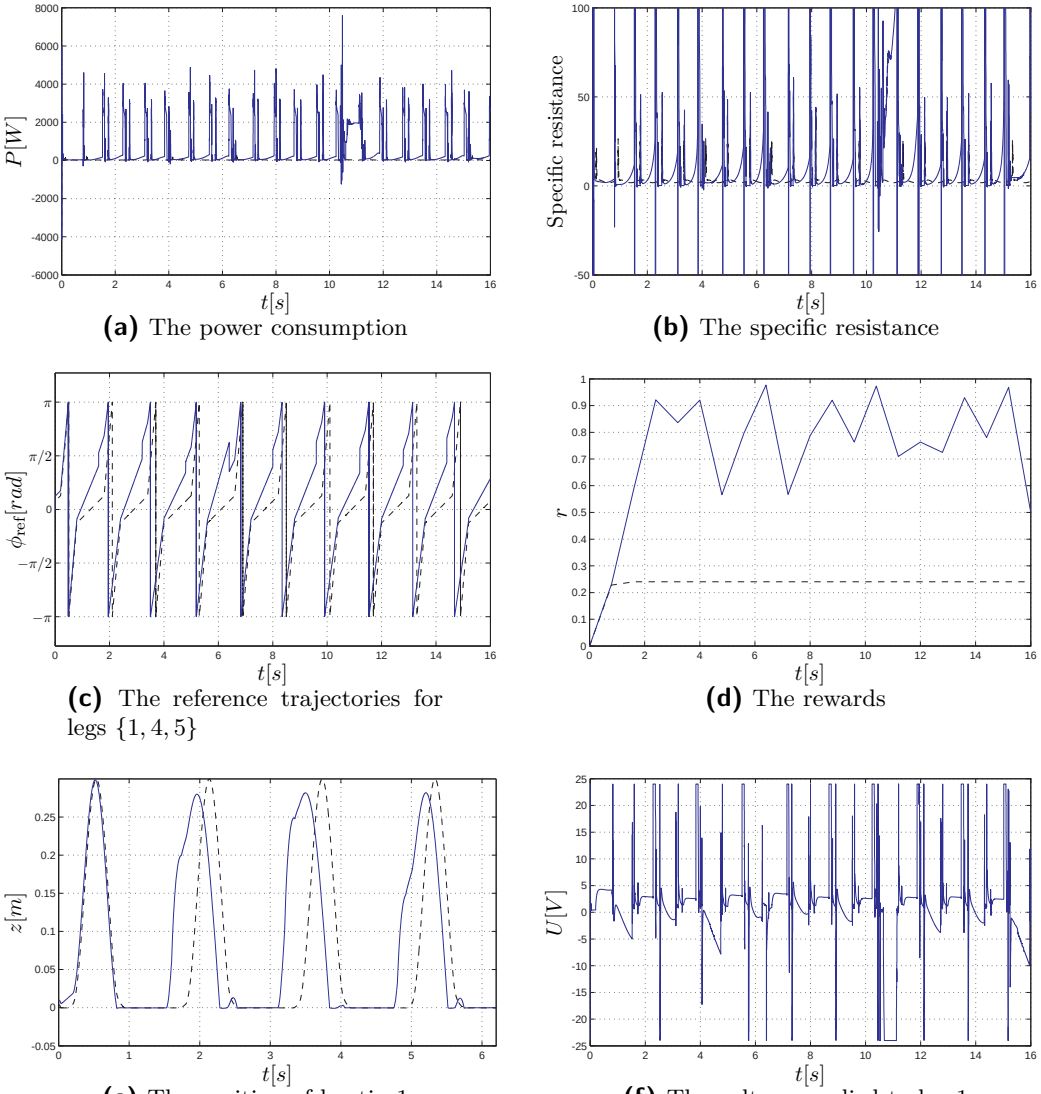
(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\bar{\phi}$ **(f)** The angular velocity $\dot{\bar{\phi}}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-11: Experiment 1 - maximizing the velocity, flat terrain. States during run in which an aerial phase is achieved. These figures show the states of the robot before (dashed black lines) and after optimization (solid blue lines). As can be seen the states vary greatly before and after optimization.



(a) The power consumption

(b) The specific resistance

(c) The reference trajectories for legs $\{1, 4, 5\}$

(d) The rewards

(e) The position of leg tip 1

(f) The voltage applied to leg 1

Figure 5-12: Experiment 1 - maximizing the velocity, flat terrain. Properties during run in which an aerial phase is achieved. These figures compare some of the properties of the system before (dashed black lines) and after (solid blue lines) optimization. As can be seen the power consumption contains very high peaks, indicating a non-smooth walk. The rewards shown are the sampled rewards, with sampling frequency $1.25\ Hz$. Furthermore, the x- versus the z-position of leg tip 1 is shown and the voltages applied to this leg are presented. Figure 5-12e shows clearly the leg lifting off and landing again.

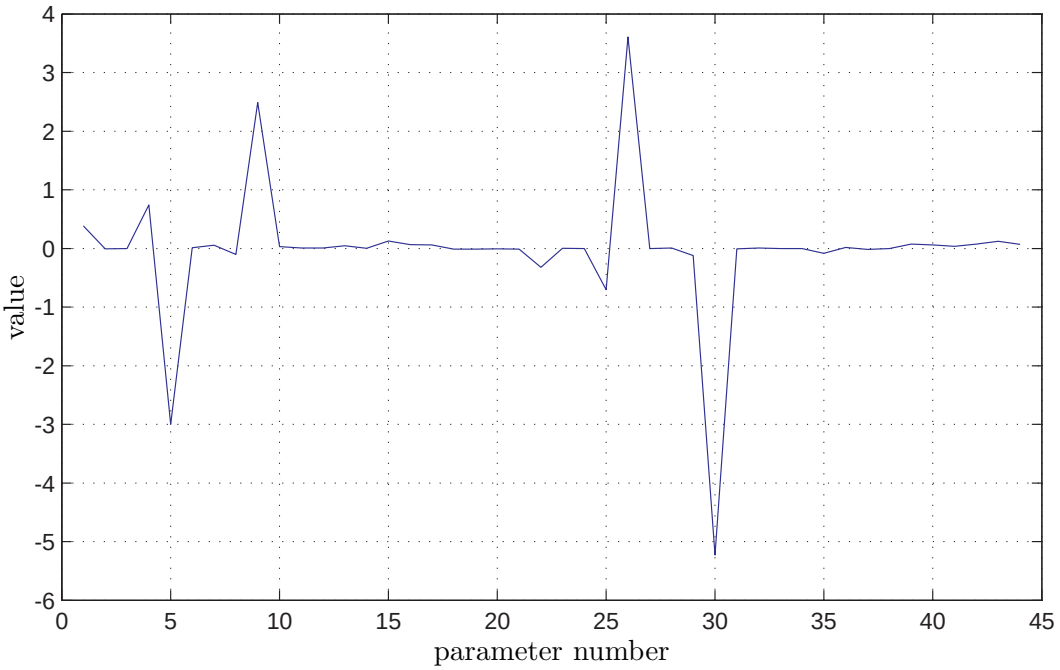
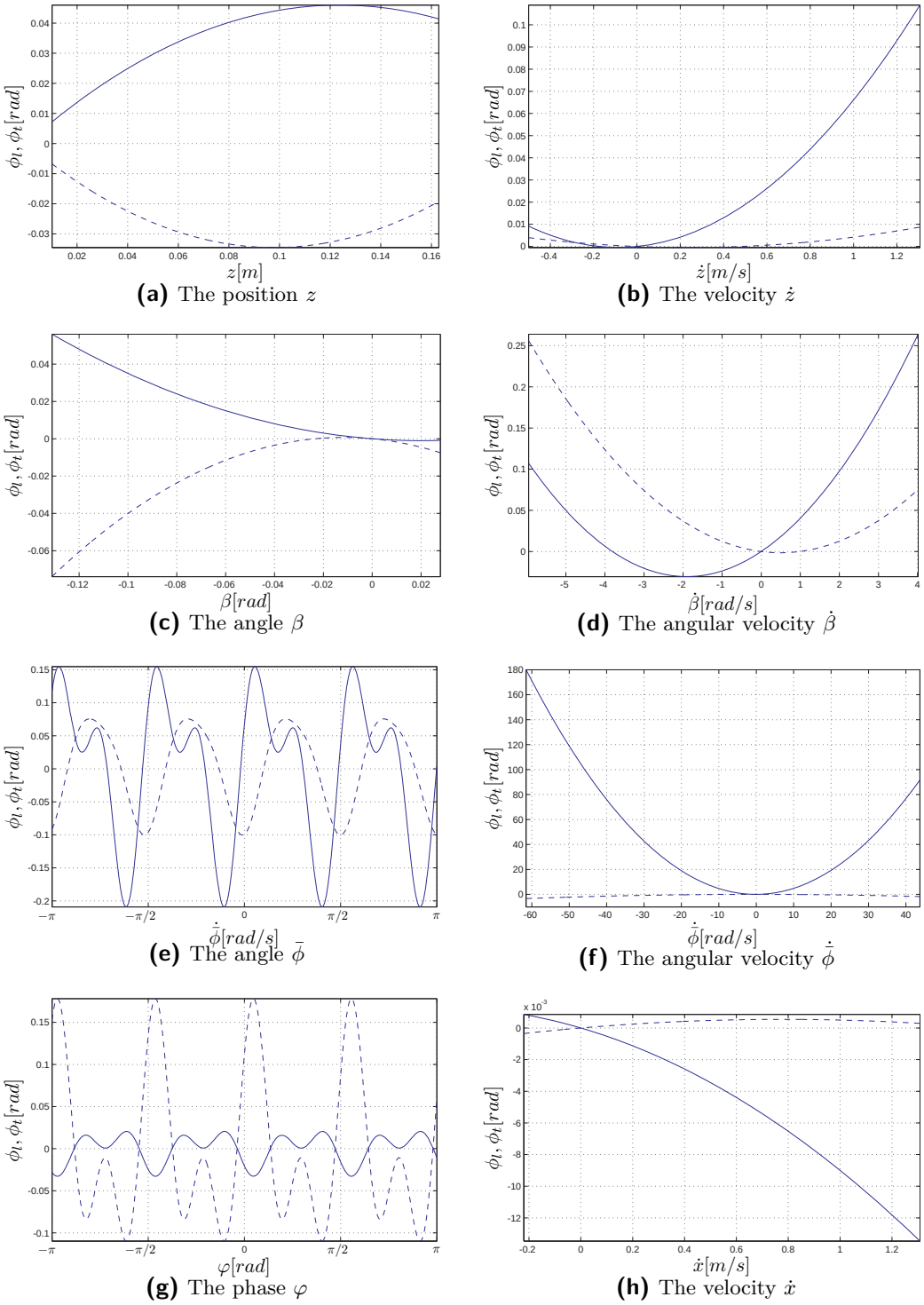


Figure 5-13: Experiment 1 - maximizing the velocity, flat terrain. Parameter values for run in which an aerial phase is achieved. To which state/policy part each number corresponds, see Table 5-5.



(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\dot{\phi}$ **(f)** The angular velocity $\dot{\phi}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-14: Experiment 1 - maximizing the velocity, flat terrain. Influence of states on actions for run in which an aerial phase is achieved. These figures show the influence of the states on the actions. The solid lines denote the lift-off angle and the dashed lines denote the touchdown angle. For the range of each state the maximum and minimum values as occurred in Figure 5-11, after optimization, are used.

5-2-3 Conclusions and discussion

In the previous section the results of Experiment 1 were presented, which will be discussed in this section, to start with the mean and standard deviation of the returns (Figure 5-3, Figure 5-4). As can be seen the mean of the return shows an increase over the parameter updates of 17% (and a maximum increase of 120%), but at the same time the standard deviation is very large and does not follow the mean, which indicates that the results differ largely. However, upon closer inspection of Figure 5-4 it can be seen that the large standard deviation is mostly caused by runs that led to a much higher return than the mean. Furthermore, only 1 run had to be discarded as it violated the requirement $\phi_t < \phi_l$. The increase in return is very large at the first or second parameter update and then plateaus. This indicates that the learning rate was relatively high, and the method finds a local optimum. The optimization was approximately converged after six parameter updates, which lasted 72 minutes of simulated time.

Second, the actions applied to the system are discussed (Figure 5-5). As expected the angle between the legs becomes larger, necessary to increase the velocity on flat terrain. After the first sample step the actions stay approximately constant, and the lift-off and touchdown angles are increased and decreased by approximately the same amount. The latter shows that the actions are mirrored with respect to each other.

Third, the states before and after optimization are analysed. As mentioned under Figure 5-6 most states remain the same under optimization of the velocity. The largest difference can be seen in the velocity $\dot{x}$, which is conform expectations. Furthermore, differences are seen in the position z and velocity $\dot{z}$, which are directly related to the change in velocity. Because there is a larger angle between lift-off and touchdown, compared to the initial situation, the body of the robot will move down further, which is accompanied by an increasing velocity in vertical direction.

In Figure 5-7 other properties were compared. From the comparison of the reference trajectories it follows directly that the angle between lift-off and touchdown increased. The power consumption increased during, and slightly after, the double stance period. This is due to the fact the legs are rotating over a larger angle when on the ground, which offers more resistance than rotating freely in the air. The specific resistance is increased a little after optimization, which is expected as the velocity is maximized but the power consumption is not minimized in any way.

Finally, the resulting parameter vectors are analysed. As can be seen from Figure 5-8a the parameters vary from parameter vector to parameter vector and do not show a clear indication of the optimal value. Therefore, in Figure 5-8b the parameter values corresponding to the run with the highest return were shown. Due to the fact that the stochastic functions in the policy are the same for both actions, it is expected there is a relation between the parameters belonging to a certain state for action ϕ_l and action ϕ_t. This expectancy is confirmed by the figure, as the peaks occur at the same parameters[14]. Furthermore, in Section 5-1 it was stated that it is expected to find

[14] This is not exactly true, as peak 15 is not mirrored in 36. However, both parameters belong to the same state.

some negative relation between the actions and the pitch of the body. This relation is confirmed by the results of the optimization as parameter 8 and 29, belonging to state β are both negative. The influence of the states on the actions is shown in Figure 5-9. It can be seen that most states do not have a large influence on the actions, except for $\bar{\phi}, \dot{\bar{\phi}}$ and φ, which are also the states that vary the most within an episode. Interesting to see is that the influence of $\dot{x}, \dot{\beta}, \dot{\bar{\phi}}$ is the same on both actions, except mirrored around 0.

Aerial phase From Figure 5-10 it is found that an aerial phase is achieved by drastically changing the lift-off angle. As follows from Figure 5-12f the voltage applied to the DC motor driving leg 1 is at its limit (24 V) at many moments in time, in order to follow the reference trajectory (Figure 5-12c). In the case of simulation this does not matter, however, when applying this parameter set to the actual robot the DC motors will wear a lot faster than under normal operation. The specific resistance of this particular run increased dramatically with respect to the initial run.

The resulting movement cannot be classified as running, but as hopping. From the results it follows that leg set 1 lifts off, touches down again and continues rotating while on the ground (the same holds for leg set 2). It is believed that lift-off is achieved by a rapid acceleration of the body, caused by the large change in reference trajectories. Analysis of the parameter values of Figure 5-13 shows again a clear relation between the parameters of ϕ_l and the parameters of ϕ_t. However, in contrast to the previously analysed parameter vector there is no negative relation between β and ϕ_l, ϕ_t, but a positive relation between β^2 and ϕ_l and a negative relation between β^2 and ϕ_t. In Figure 5-14 the state $\dot{\bar{\phi}}$ seems to have a very large influence on the action, but that is because its extreme values are very large. The influence of $\dot{\beta}, \dot{\bar{\phi}}$ and φ is approximately the same. Furthermore, the influence of z and β on the actions is mirrored around 0.

The final interesting result from this run is that the difference between the lift-off and touchdown angle is approaching π rad. In the reference trajectory this manifests itself as an almost perfect sawtooth, compared to the initial reference trajectory. The movement of the robot is thus approaching the movement of the robot in the determination of the reference trajectory tracker parameters (Figure 4-6), which was considered to be the extreme case.

Final conclusion The eNAC method does find some good and expected results, but it does not show a clear increase in return in every optimization performed. It is believed that this has to do with the method only capable of finding a local optimum and the system being largely deterministic; only the actions and starting states are stochastic and there are no external disturbances, causing the robot to move in a very stable manner. Due to this there is only a slight variation in states between samples taken within an episode, and even between episodes in the same parameter update. This makes it very difficult for the eNAC method to determine the natural gradient, and thus the parameter updates. Therefore, in Section 5-3 varying ground height is considered.

5-3 Experiment 2 - maximizing the velocity, varying ground height

The second experiment follows from the conclusions drawn in Section 5-2-3. Again the velocity of the robot in x-direction is maximized, but a varying ground height is used in this experiment. This is done in order to enter disturbances into the system, with the intention to find better results than with a completely flat terrain. The experiment specific settings are discussed in Section 5-3-1 and an overview of the results is given in Section 5-3-2. Finally, in Section 5-3-3 the results are discussed and conclusions are drawn.

5-3-1 Experiment specific settings

An overview of experiment specific settings is given in Table 5-6. The ground profile is shown in Figure 5-15, and is the same in all runs. In Figure 5-17 an overview is given of the states before optimization. As can bee seen there is a larger variation in state values during an episode than with Experiment 1. Furthermore, the moments at which the legs are lifting off from, or touching down on the ground are less well defined because of the changing ground height. Combining this leads to the use of a sampling rate of 3.75 Hz, which equals taking six samples per cycle time[15]. The simulation time is 16 s, which is necessary in order for the robot to walk over all the obstacles in Figure 5-15. The learning rate of the actor is kept constant with respect to Experiment 1, thus at 0.025. The number of runs is limited to only 5 due to time constraints[16].

Table 5-6: Settings Experiment 2

Parameter	Symbol	Value
Weighted average weight	α	1
Sampling rate		3.75 Hz
Simulation time		16 s
Learning rate actor	α_a	0.025
Number of runs		5

Plots of the power consumption, the specific resistance, the reference trajectories of the legs and the rewards are shown in Figure 5-16. From the plot of the power consumption it can be clearly seen that the power consumption, and consequently the specific resistance, rises when the robot has to walk over an obstacle, caused by having to lift its body over a larger distance. The initial average return, using no exploration in the policy, is 0.3276, which corresponds to an initial average velocity of 0.12 m/s. The specific resistance of the robot is given by 6.02, comparable to "Big Huskie" shown in Figure 1-2.

[15] There have been six runs performed with the sampling rate raised to 6.25 Hz, i.e. ten samples per cycle time. Results do not show any improvement or deterioration of the return.

[16] A complete run costs over 8 hours of simulation time to complete.

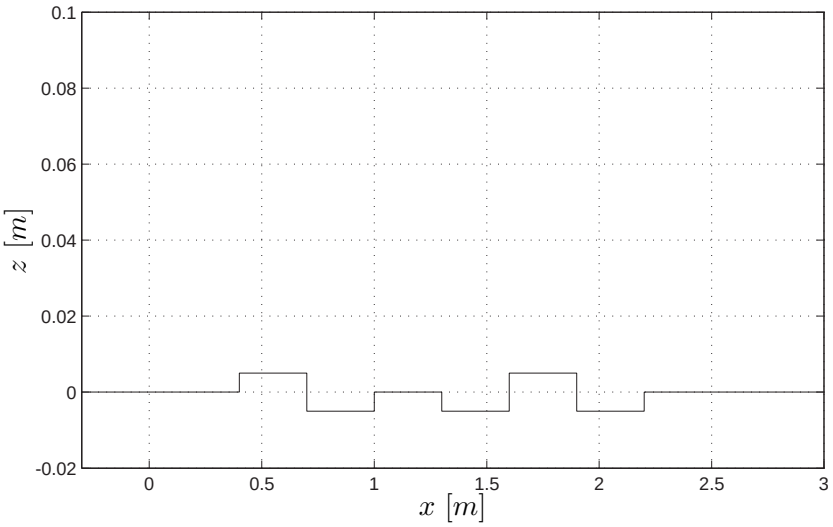
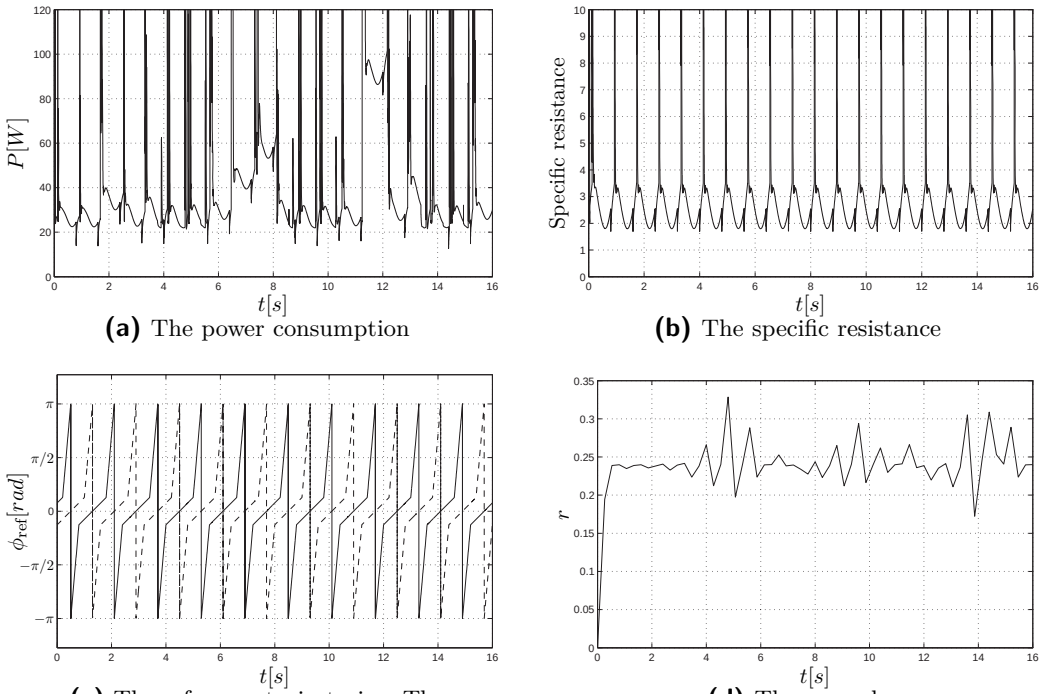


Figure 5-15: Experiment 2 - maximizing the velocity, varying ground height. Ground profile. The maximum instantaneous change of the ground profile is $0.01\ m$, which corresponds to approximately 7% of the leg length.



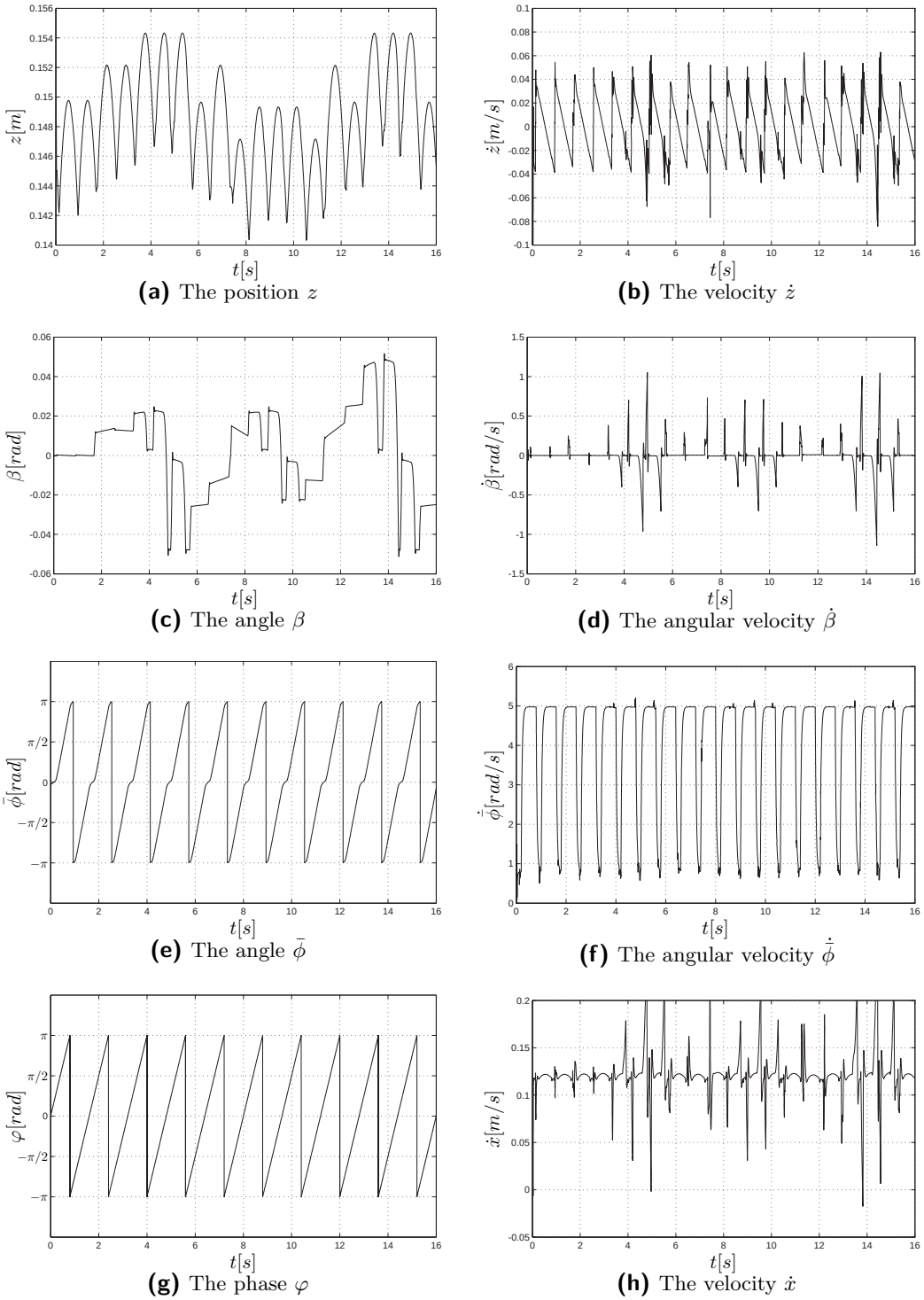
(a) The power consumption

(b) The specific resistance

(c) The reference trajectories. The solid line denotes the reference trajectory for legs $\{1, 4, 5\}$, the dashed line denotes the reference trajectory for legs $\{2, 3, 6\}$.

(d) The rewards

Figure 5-16: Experiment 2 - maximizing the velocity, varying ground height. Initial properties. These figures show some of the properties before optimization. The plots of the power consumption and specific resistance are zoomed in and do not show the entire range. The rewards shown are the sampled rewards, with sampling frequency $3.75\ Hz$.



(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\bar{\phi}$ **(f)** The angular velocity $\dot{\bar{\phi}}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-17: Experiment 2 - maximizing the velocity, varying ground height. Initial states. These figures show the states of the robot before optimization, using the initial policy as described in Section 5-1, but exploration turned off. As can be seen there is a larger variation in state values during an episode than with Experiment 1.

5-3-2 Results

There were only 5 runs performed within this experiment and therefore there are no plots shown of the mean and standard deviation. The returns of 4 runs are shown in Figure 5-4; one run is discarded due to violating the requirement $\phi_l < \phi_t$. The return increases on average from 0.2376 to 0.2813, which belongs to an increase of the average velocity from 0.12 m/s to 0.14 m/s.

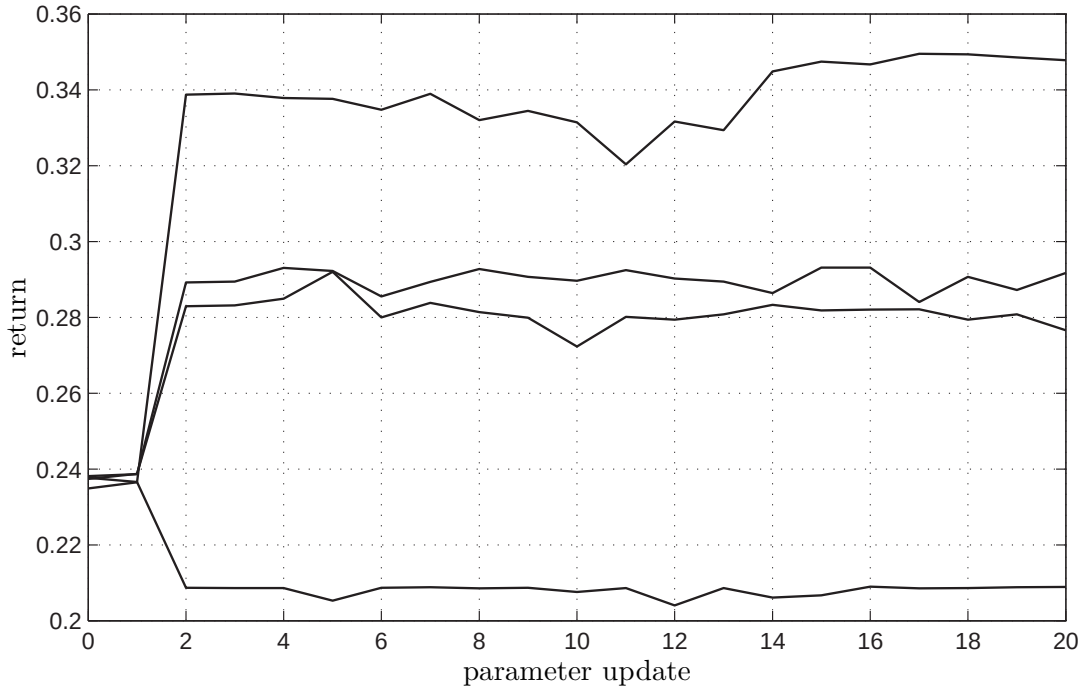


Figure 5-18: Experiment 2 - maximizing the velocity, varying ground height. Returns. This figure shows the (intermediate) average returns of all episodes within a parameter update.

Each run consisted of 20 parameter updates, and the average number of episodes in a run is found to be 1006. Considering 10 complete leg cycles per episode this gives a total simulated time of 16096 s or approximately 4.5 hours. However, as can be seen from Figure 5-4 all runs approximately converged within 3 parameter updates, reducing the average number of episodes to 184, which equals 2944 s or 50 minutes per run. Furthermore, the robot walked for approximately 2 meters per episode, giving a total walking distance of roughly 370 meters to find the optimal result. The specific resistance before optimization was given by 6.02 and after optimization it is found to be 19.70.

Comparing the situations before and after optimization

To compare the behaviour of the robot before optimization to the behaviour after optimization, use is made of the run with the highest return. The actions are shown in Figure 5-19, in Figure 5-20 the states of the system are compared and in Figure 5-21 properties such as the power consumption, the specific resistance, the reference trajectories of the legs and the rewards are compared.

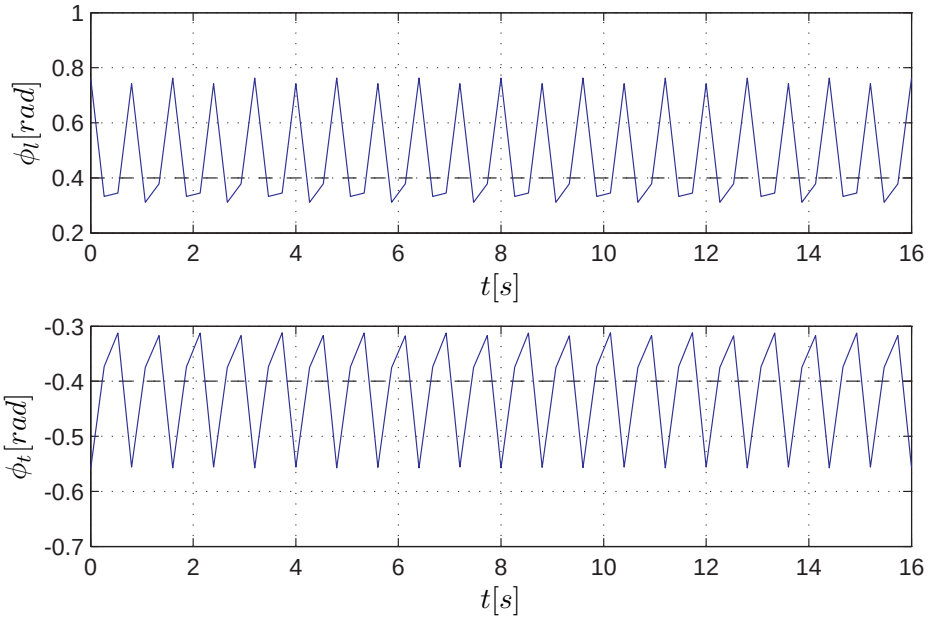
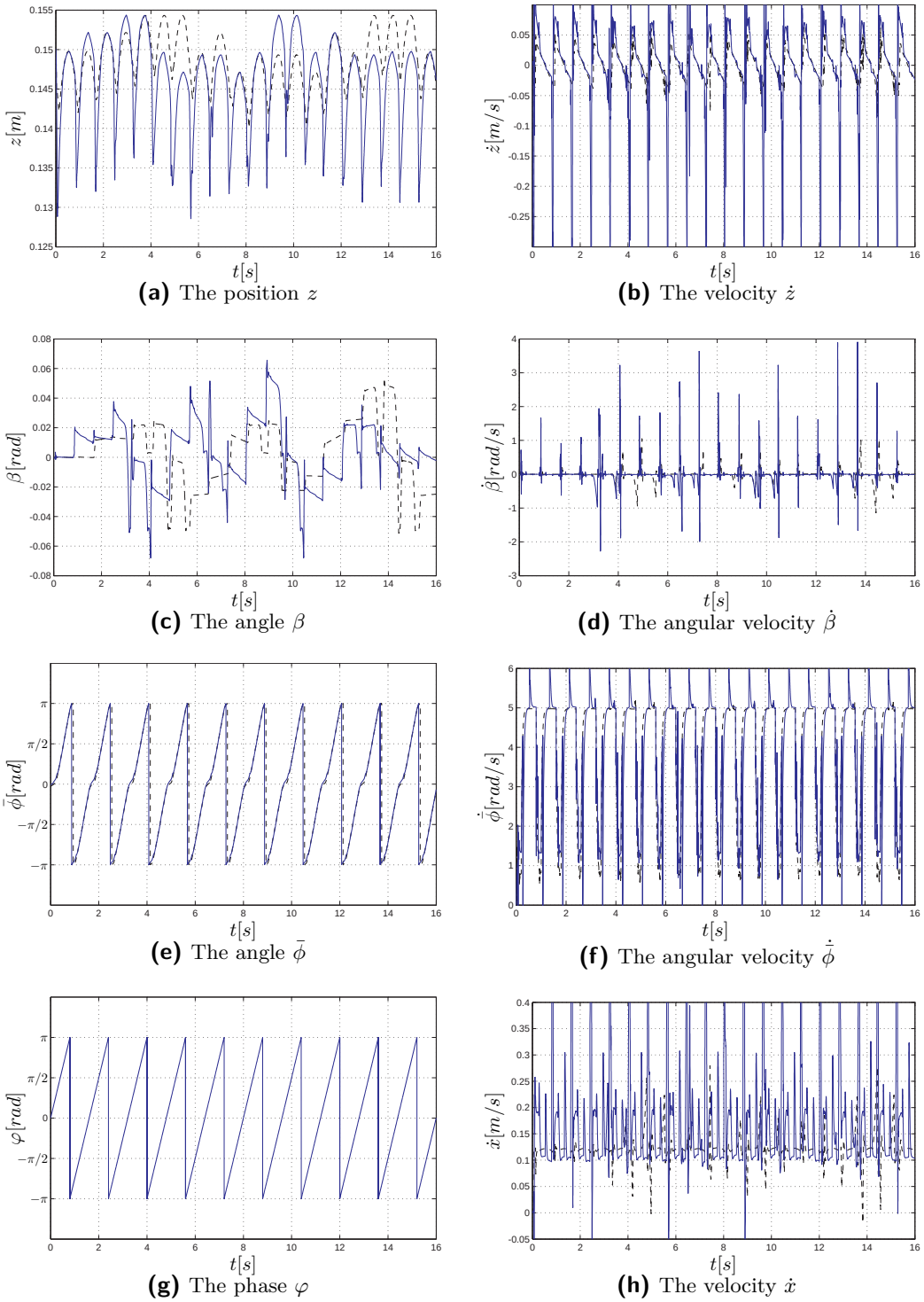


Figure 5-19: Experiment 2 - maximizing the velocity, varying ground height. Comparing the actions before and after optimization. This figure compares the actions before optimization (dashed black lines) to the actions after optimization (solid blue lines).



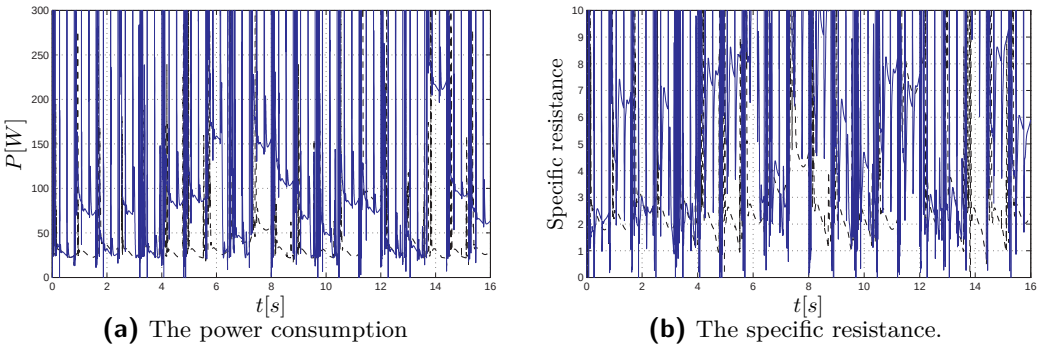
(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\bar{\phi}$ **(f)** The angular velocity $\dot{\bar{\phi}}$

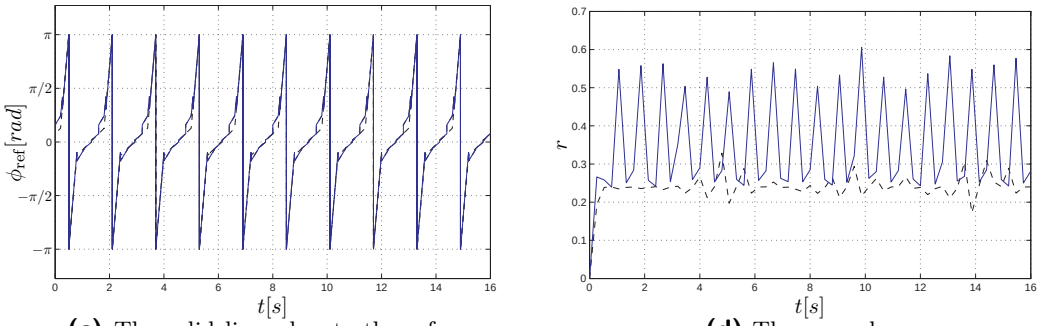
(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-20: Experiment 2 - maximizing the velocity, varying ground height. Comparing the states before and after optimization. These figures show the states of the robot before (dashed black lines) and after optimization (solid blue lines).



(a) The power consumption

(b) The specific resistance.



(c) The solid lines denote the reference trajectories for legs $\{1, 4, 5\}$, the dashed lines denote the reference trajectories for leg sets $\{2, 3, 6\}$.

(d) The rewards

Figure 5-21: Experiment 2 - maximizing the velocity, varying ground height. Comparing properties before and after optimization. These figures compare some of the properties of the system before (dashed black lines) and after (solid blue lines) optimization. The plots of the power consumption and specific resistance are zoomed in and do not show the entire range. The rewards shown are the sampled rewards, with sampling frequency $3.75\ Hz$.

Influence of the parameters on the policy

The policy parameters obtained after optimization are presented in Figure 5-22. It is chosen only to show those belonging to the run with the highest return. The meaning of the numbers is shown in Table 5-5. The influence of each state on the actions is given in Figure 5-23. For the range of each state the maximum and minimum values as occurred in Figure 5-20, after optimization, are used.

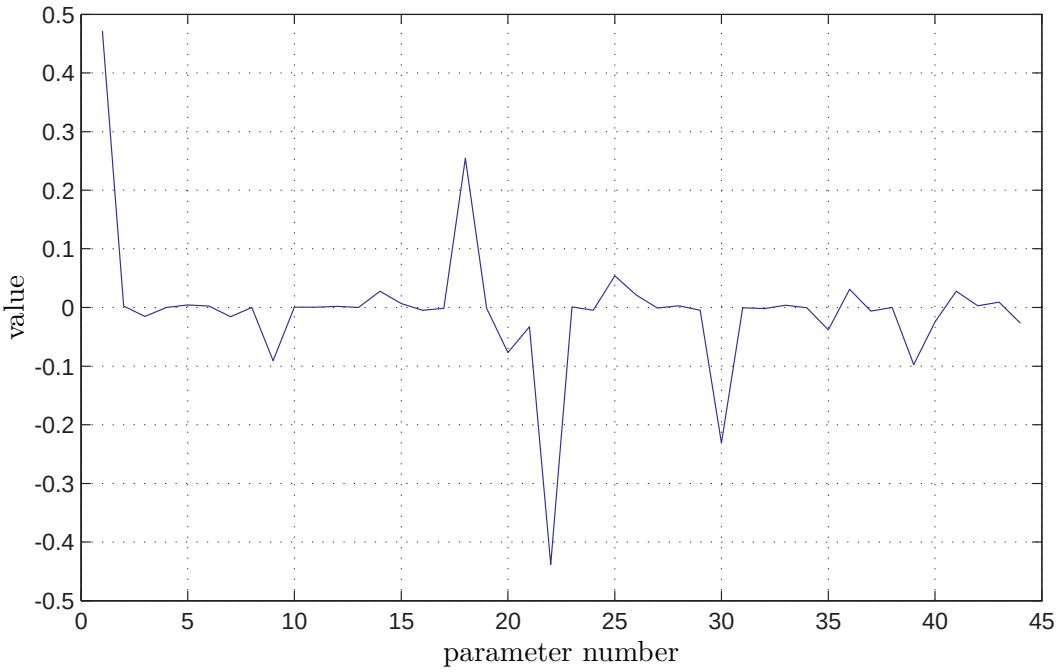
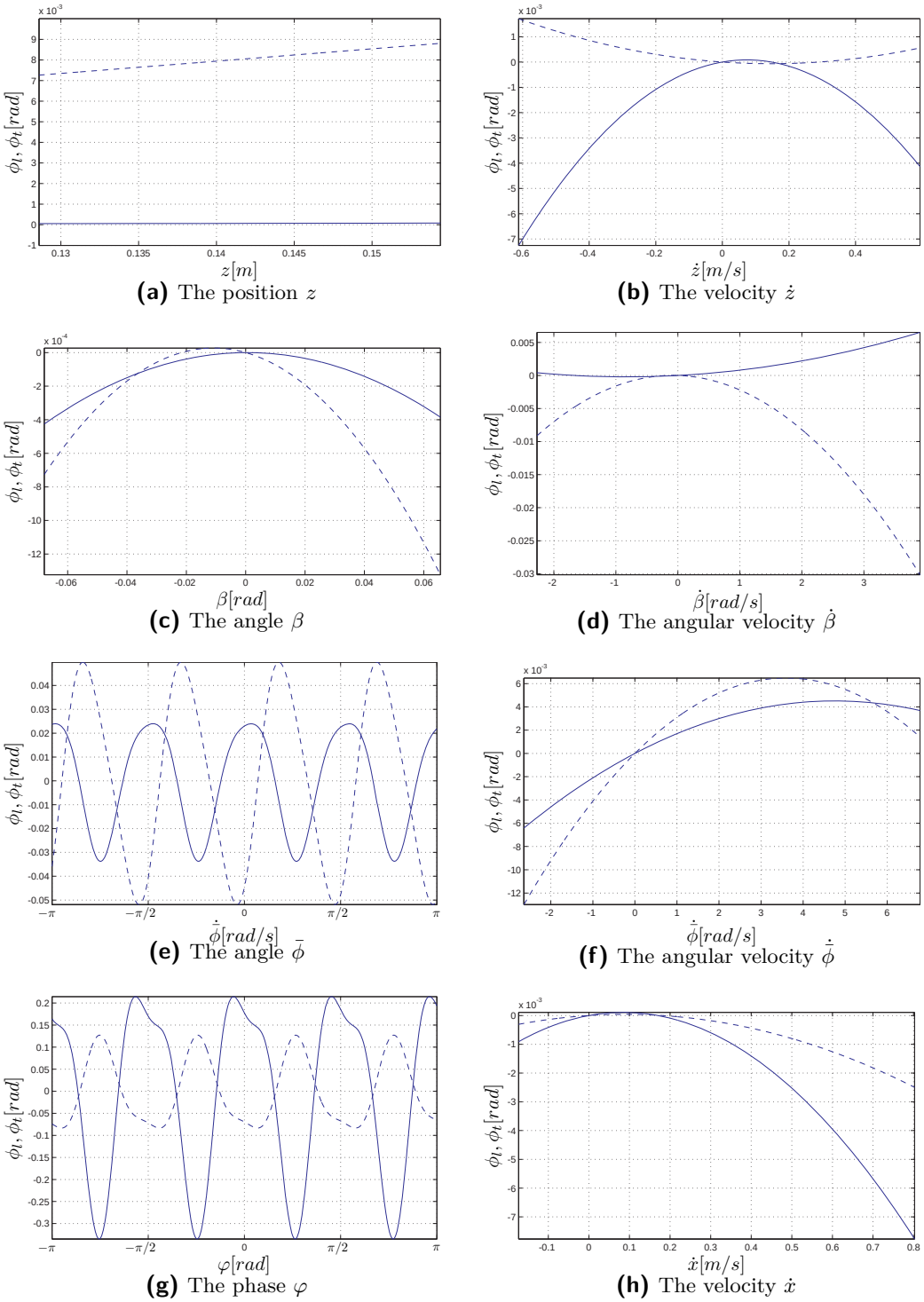


Figure 5-22: Experiment 2 - maximizing the velocity, varying ground height. Parameter values of the run with the highest return after optimization. To which state/policy part each number corresponds, see Table 5-5.



(a) The position z **(b)** The velocity $\dot{z}$

(c) The angle β **(d)** The angular velocity $\dot{\beta}$

(e) The angle $\dot{\phi}$ **(f)** The angular velocity $\dot{\phi}$

(g) The phase φ **(h)** The velocity $\dot{x}$

Figure 5-23: Experiment 2 - maximizing the velocity, varying ground height. Influence of states on actions. These figures show the influence of the states on the actions. The solid lines denote the lift-off angle and the dashed lines denote the touchdown angle. For the range of each state the maximum and minimum values as occurred in Figure 5-20, after optimization, are used.

5-3-3 Conclusions and discussion

As follows from Figure 5-18, 3 out of 4 experiments led to an increase in return, on average of 18% (comparable to the increase found in Experiment 1), and the maximum increase found was by 46%. One run violated the requirement $\phi_t < \phi_l$ and was therefore discarded. Because only five runs were considered it is difficult to say with confidence that the eNAC method can optimize the problem as described in Chapter 4, with varying ground height, but the results point in that direction. The optimization approximately converged after 3 parameter updates, which lasted 50 minutes of simulated time. The fast increase in return and plateauing afterwards indicate a high learning rate and the method finding a local optimum. The specific resistance before optimization was 6.02 and after optimization 19.70, from which it follows that an increase in velocity goes together with an even higher increase in power consumption.

The actions applied to the system vary between 0.35 rad to 0.7 rad for the lift-off angle and -0.55 rad to -0.3 rad for the touchdown angle. Upon closer inspection it is found that the difference between angles is smallest when one leg set just lifted off and the difference is largest right before touchdown of this leg set. The lift-off angle increased by a larger amount than the touchdown angle decreased, which is apparently necessary to walk over the specific obstacles used.

Figure 5-20a confirms that the robot is indeed walking faster after optimization as the change in body angle is shifted to the left (i.e. rotations happen earlier in time). Furthermore, the velocity in x-direction, the height of the robot and the velocity in z-direction change by a relatively large amount during optimization, but the remaining states stay approximately the same.

The reference trajectory changes significantly when a leg is on the ground, causing the leg to accelerate and to quickly increase the step size of the robot. The sudden increase in reference trajectory is clearly visible in Figure 5-21a where an increase in power consumption is shown at the same time. Furthermore, the power consumption rises at the moments the robot has to walk over an obstacle, caused by lifting up his body over a larger distance.

By analysing the parameter vector found in the run with the highest return, a clear negative relation is found between the pitch angle and the lift-off and touchdown angles. To be more precise, the squared pitch angle is of importance rather than just the pitch angle. The influence of the states on the actions is shown in Figure 5-23, where it can be seen that many states do not have a large influence, except for $\bar{\phi}$ and φ, which are also the states that vary most. The influence of z and $\dot{\beta}$ on the actions is mirrored around 0.

Final conclusion The eNAC method is capable of optimizing the velocity of the robot while it is walking over terrain with varying ground height, although the results are not 100% conclusive. It is believed this is caused by the fact the method is only capable of finding a local optimum in combination with a large learning rate. Due to a large variation of state values within an episode a sampling rate of 3.75 Hz is employed.

However, using a higher sampling rate of 6.25 Hz does not yield an improvement in the return.

5-4 Conclusions and discussion

The eNAC method showed success in optimizing the velocity, on flat terrain as well as on terrain with a varying ground height, where the robot is walking with a tripod gait. On average an increase in return was found of approximately 20% for both experiments. Furthermore, the maximum increase in return found for flat terrain was 120% and for varying ground height 46%. The optimizations for flat terrain took on average 72 minutes of simulated time and the optimizations for a varying ground height 50 minutes. The difference is believed to be caused by a larger change in state values when walking over terrain with varying ground height. With a varying ground height and a higher sampling rate used in Experiment 2, 3 out of 5 runs resulted in an increase of at least 10% in return compared to 13 out of 30 runs with Experiment 1. This indicates that the method works better when the states of the robot are varying more between samples, and more samples per cycle time are used[17].

One of the runs on flat terrain resulted in hopping behaviour of the robot, i.e. the lift-off angle would change drastically causing the robot to be launched in the air and landing on the same legs again. The resulting average velocity increased over 200%. Unfortunately, the body would be touching the ground during walking thus this run was discarded.

The resulting policy parameters showed that the influence of the periodic states on the policy is relatively large, which is believed to be caused by the fact that these state values vary most within an episode. It was observed that this influence directly caused the robot to walk faster over the varying ground by lifting up its leg just before an obstacle. This indicates that modelling the influence of periodic states on the actions by Fourier functions indeed is possible. Furthermore, it was shown for some states, especially the angular velocity of the body, that the influence on both actions is equal, but mirrored around 0.

The specific resistance showed an increase in both experiments, indicating there is no linear relation between the velocity and the power consumption, where the specific resistance in Experiment 2 showed the largest relative increase. In the previous sections it was not listed, but experiments have been performed to optimize the specific resistance and a weighted average of velocity and power consumption (with weighting factor $\alpha = 0.5$). However, optimizing for specific resistance failed without exception; the requirement $\phi_t < \phi_l$ was violated or the robot would fall over. It is believed the source of this lies in the very large peaks in power consumption, as shown in e.g. Figure 5-7a, combined with the fact that a negative value for velocity or power consumption would

[17] This holds up to a certain number of samples; results show that a large sampling rate does not yield success, believed to be caused by the large number of actions that do not have an influence on the return but act more or less as random noise.

lead to a positive return[18]. However, it is only possible to modify the reward function of the specific resistance in some kind of ad-hoc manner, as negative values can occur during optimization runs due to the stochastic nature of action selection. This makes it that the specific resistance is not an effective reward function for use in this specific problem. Optimization runs using the weighted average showed similar behaviour, where the method was not capable of optimizing the return. A possible solution to the problem is using another type of controller, which can avoid the large peaks in power consumption.

A final conclusion is made regarding the gait generation method, and especially the way the reference trajectories are generated, in combination with the structure of the optimization problem. The reference trajectories are generated based on the timings of lift-off and touchdown events and the lift-off and touchdown angles. The latter are the actions applied to the system, of which the underlying structure was optimized. Both actions were changed at the same time, causing the reference trajectories to change according to the new lift-off and touchdown angles. It might be better to change only the touchdown angle for the leg that is in the air, fixing the previously used lift-off angle (and similar for the leg that is on the ground). This is motivated by the fact the current position of the leg should be determined by the previously occurred lift-off angle and the desired touchdown angle, rather than by a lift-off angle that never took place.

[18] In the way the reward function for the specific resistance is defined, the better the results the closer the return to 0^{-1}. A small negative velocity, that is not accompanied by a negative power consumption, would give a very large reward although the behaviour is far from optimal.

Chapter 6

Conclusions

In this chapter a short overview is given of the most important results derived in this thesis. Furthermore, future work is listed and the final conclusions are presented.

Modelling of legged systems

A model of Zebro has been developed, based on the Euler-Lagrange equations. This model was not intended to be a perfect model of Zebro, due to some crude assumptions on the geometry of the robot, e.g. the semi-circular legs are modelled as straight rigid bodies. Therefore, this model was only used for qualitative analysis not for quantitative analysis, i.e. parameters found during learning are not interchangeable between the model and Zebro. The ground contact was modelled as a system of springs and dampers, giving a trivial implementation into the model.

The gait generation used was the Switching Max-Plus-linear model, which is a discrete event method of modelling legged locomotion based on two events: the lift-off and touchdown of legs. The reference trajectory for the legs to follow was based on the leg angles at the lift-off and touchdown events.

The model of Zebro was implemented, together with the gait generation method, in Matlab/Simulink in order to test the learning method on.

The eNAC method

In order to learn the optimal gait parameters, with respect to velocity and power consumption, the episodic Natural Actor-Critic (eNAC) method was applied. This is a model-free Actor-Critic (AC) method that applies the natural gradient in order to update the parameters of the policy. The advantage of this method over other Reinforcement Learning (RL) methods found in literature is that it does not require a set

of user-defined basis functions to approximate the value function, making it a suitable method to apply to high-dimensional, continuous state spaces. Furthermore, the episodic nature of this method makes it particularly suitable for the problem of gait optimization in which clear episodes can be defined.

Learning the optimal gait parameters

In order to reduce the number of states present in the system the concept of a virtual leg was introduced, which is an average of all legs present in the system. The actions applied to the system were the lift-off and touchdown angles for the legs. These actions were generated using a policy which was based on the states present in the system and the periodic states were incorporated using a Fourier approximation. Results show that the concept of a virtual leg can be applied even under varying ground height. Furthermore, the use of Fourier approximation of the periodic states directly led to an increase in return.

Two different experiments were performed, both with the goal of maximizing the velocity. However, one experiment was performed with the robot walking on flat terrain and the other with the robot walking on terrain with varying ground height. Both experiments resulted in an average increase in velocity of approximately 20%, with a maximum increase of 120% and 50% for flat terrain and terrain with varying ground height, respectively. The average simulated time to find these results was 60 minutes. Moreover, one optimization run resulted in a hopping gait with which the robot was able to move more than three times as fast as with the initial parameter set, at the cost of a much higher power consumption.

The results showed furthermore a clear relation between parameters that belong to a certain state but different actions. However, results did not show a clear relation between parameter values of different optimization runs. A source of this is believed to be the eNAC method only capably of finding a local optimum. It was found that the states that have the largest influence on the actions also had the largest range in state values.

Future work

As explained in Chapter 5 there is very little intuition on how to choose a correct policy in terms of which states to include, with which order function to approximate their influence and whether or not to include a bilinear part. This lead to the decision to include all states, and approximate them with the same order functions, giving a relatively high number of parameters. Therefore it must be investigated if techniques can be applied that shape the policy such that the influence of the states are better incorporated, possibly reducing the number of parameters and/or improving the result.

In order to determine the gradient of the policy parameters use was made of simple linear regression methods. However, the use of more sophisticated methods for linear

regression should be investigated in order to increase the accuracy of the gradient estimate and speed up the learning process.

It was not possible to optimize the gait for the specific resistance or a weighted average between velocity and power consumption. This had to do with large spikes in power consumption inherent to the problem of impact. The reference trajectory tracker applied was a simple PD-controller, not capable of actively controlling the impact on the ground. Applying e.g. zero-torque control can prevent large spikes in the power consumption making it possible to include it in the optimization process.

Furthermore, a different implementation of the continuous time scheduler must be considered. At the moment both the lift-off and touchdown angle of a leg are changed at the same time. However, it is believed that only changing the angle at the next event, i.e. changing only the touchdown angle when a leg is in the air, will lead to better results. To go even one step further, a complete redesign of the continuous time scheduler is suggested. Take for example the situation in which the leg is in the air and the lift-off angle is left constant. If the touchdown angle is suddenly made smaller the current continuous time scheduler will generate a trajectory that forces the leg to rotate back. However, this is not optimal in any sense as this will lead to high voltages to control the position of the leg, and thus increase power consumption. Furthermore, rotating the legs back causes the Centre of Mass (CoM) of the body to move backwards. This negative velocity will influence the specific resistance rendering it useless. Instead, the leg should be slowed down and continue moving towards the touchdown angle. This mechanism will allow for a smoother change in leg position.

Finally, it was not possible to investigate how the eNAC method performs on the real-life set-up. This is a far more challenging situation due to possible limited state information and random, external disturbances such as the ground height.

Final conclusions

In Chapter 1 the question was posed if the eNAC method is capable of learning the optimal gait parameters for a hexapod robot with respect to velocity and power consumption, where the gait is generated using the Switching Max-Plus-linear model, using the eNAC method. This question can be answered positively as well as negatively. Yes, the method learned parameter values that led to an increase in velocity of up to 120%. No, the method was not capable of reducing the power consumption. However, it is believed that by further investigating the points given under future work, the power consumption can be included within the optimization. More importantly, by applying the necessary changes to the reference trajectory generation it is expected to find better results. Therefore, we are confident that the eNAC method can be applied to the real-life set-up with success.

Appendix A

Equations of motion

In this appendix the matrices and vectors as they appear in the equations of motion given by

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + G(\boldsymbol{q}) = \boldsymbol{u} + \boldsymbol{J}(\boldsymbol{q})^T\boldsymbol{\tau}$$

are listed. The external forces and inputs are not listed here.

Matrix $M(\boldsymbol{q}$

See Eq. (A-3).

Matrix $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$

See Eq. (A-4).

Vector $G(\boldsymbol{q})$

See Eq. (A-1).

$$G(\boldsymbol{q}) = \begin{bmatrix} 0 \\ g\left(m_b + 6m_{\text{leg}}\right) \\ -gl_{\text{leg}}m_{\text{leg}}\left(\sum_{i=1}^{6}\left(-\sin\left(\beta - \phi_i\right)\right)\right) \\ -gl_{\text{leg}}m_{\text{leg}}\sin\left(\beta - \phi_1\right) \\ -gl_{\text{leg}}m_{\text{leg}}\sin\left(\beta - \phi_2\right) \\ -gl_{\text{leg}}m_{\text{leg}}\sin\left(\beta - \phi_3\right) \\ -gl_{\text{leg}}m_{\text{leg}}\sin\left(\beta - \phi_4\right) \\ -gl_{\text{leg}}m_{\text{leg}}\sin\left(\beta - \phi_5\right) \\ -gl_{\text{leg}}m_{\text{leg}}\sin\left(\beta - \phi_6\right) \end{bmatrix} \tag{A-1}$$

Change of coordinates

See Eq. (A-2).

$$H(\boldsymbol{q}) = \begin{bmatrix} x \\ z \\ \beta \\ l_{\text{leg}} \sin(\beta - \phi_1) + \cos(\beta) p_{\text{leg}} + x \\ -l_{\text{leg}} \cos(\beta - \phi_1) + \sin(\beta) p_{\text{leg}} + z \\ \beta + \phi_1 \\ l_{\text{leg}} \sin(\beta - \phi_2) + \cos(\beta) p_{\text{leg}} + x \\ -l_{\text{leg}} \cos(\beta - \phi_2) + \sin(\beta) p_{\text{leg}} + z \\ \beta + \phi_2 \\ l_{\text{leg}} \sin(\beta - \phi_3) + x \\ z - l_{\text{leg}} \cos(\beta - \phi_3) \\ \beta + \phi_3 \\ l_{\text{leg}} \sin(\beta - \phi_4) + x \\ z - l_{\text{leg}} \cos(\beta - \phi_4) \\ \beta + \phi_4 \\ l_{\text{leg}} \sin(\beta - \phi_5) - \cos(\beta) p_{\text{leg}} + x \\ -l_{\text{leg}} \cos(\beta - \phi_5) - \sin(\beta) p_{\text{leg}} + z \\ \beta + \phi_5 \\ l_{\text{leg}} \sin(\beta - \phi_6) - \cos(\beta) p_{\text{leg}} + x \\ -l_{\text{leg}} \cos(\beta - \phi_6) - \sin(\beta) p_{\text{leg}} + z \\ \beta + \phi_6 \end{bmatrix} \tag{A-2}$$

Jacobian of change of coordinates

See Eq. (A-5).

$$
M(q) = \begin{bmatrix}
m_{\text{body}} + 6m_{\text{leg}} & 0 & \left(\sum_{i=1}^{6} \cos(\beta - \phi_i)\right) l_{\text{leg}} m_{\text{leg}} & \cdots \\
0 & m_{\text{body}} + 6m_{\text{leg}} & -\left(\sum_{i=1}^{6} -\sin(\beta - \phi_i)\right) l_{\text{leg}} m_{\text{leg}} & \cdots \\
\left(\sum_{i=1}^{6} \cos(\beta - \phi_i)\right) l_{\text{leg}} m_{\text{leg}} & -\left(\sum_{i=1}^{6} -\sin(\beta - \phi_i)\right) l_{\text{leg}} m_{\text{leg}} & 2m_{\text{leg}} l_{\text{leg}} \left(3l_{\text{leg}} - \sin(\phi_1) p_{\text{leg}} - \sin(\phi_2) p_{\text{leg}} + \sin(\phi_5) p_{\text{leg}} + \sin(\phi_6) p_{\text{leg}}\right) + 4m_{\text{leg}} p_{\text{leg}}^2 + J_b + 6J_l & \cdots \\
-\cos(\beta - \phi_1) l_{\text{leg}} m_{\text{leg}} & -\sin(\beta - \phi_1) l_{\text{leg}} m_{\text{leg}} & -m_{\text{leg}} l_{\text{leg}}^2 + \sin(\phi_1) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & \cdots \\
-\cos(\beta - \phi_2) l_{\text{leg}} m_{\text{leg}} & -\sin(\beta - \phi_2) l_{\text{leg}} m_{\text{leg}} & -m_{\text{leg}} l_{\text{leg}}^2 + \sin(\phi_2) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & \cdots \\
-\sin(\beta - \phi_3) l_{\text{leg}} m_{\text{leg}} & J_l - l_{\text{leg}}^2 m_{\text{leg}} & 0 & \cdots \\
-\sin(\beta - \phi_4) l_{\text{leg}} m_{\text{leg}} & J_l - l_{\text{leg}}^2 m_{\text{leg}} & 0 & \cdots \\
-\sin(\beta - \phi_5) l_{\text{leg}} m_{\text{leg}} & -m_{\text{leg}} l_{\text{leg}}^2 - \sin(\phi_5) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & 0 & \cdots \\
-\sin(\beta - \phi_6) l_{\text{leg}} m_{\text{leg}} & -m_{\text{leg}} l_{\text{leg}}^2 - \sin(\phi_6) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & 0 & \cdots
\end{bmatrix}
$$

$$
\begin{bmatrix}
\cdots & -\cos(\beta - \phi_1) l_{\text{leg}} m_{\text{leg}} & -\cos(\beta - \phi_2) l_{\text{leg}} m_{\text{leg}} & -\cos(\beta - \phi_3) l_{\text{leg}} m_{\text{leg}} & \cdots \\
\cdots & -\sin(\beta - \phi_1) l_{\text{leg}} m_{\text{leg}} & -\sin(\beta - \phi_2) l_{\text{leg}} m_{\text{leg}} & -\sin(\beta - \phi_3) l_{\text{leg}} m_{\text{leg}} & \cdots \\
\cdots & -m_{\text{leg}} l_{\text{leg}}^2 + \sin(\phi_1) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & -m_{\text{leg}} l_{\text{leg}}^2 + \sin(\phi_2) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & J_l - l_{\text{leg}}^2 m_{\text{leg}} & \cdots \\
\cdots & m_{\text{leg}} l_{\text{leg}}^2 + J_l & 0 & 0 & \cdots \\
\cdots & 0 & m_{\text{leg}} l_{\text{leg}}^2 + J_l & 0 & \cdots \\
\cdots & 0 & -\cos(\beta - \phi_3) l_{\text{leg}} m_{\text{leg}} & m_{\text{leg}} l_{\text{leg}}^2 + J_l & \cdots \\
\cdots & 0 & -\cos(\beta - \phi_4) l_{\text{leg}} m_{\text{leg}} & 0 & \cdots \\
\cdots & 0 & -\cos(\beta - \phi_5) l_{\text{leg}} m_{\text{leg}} & 0 & \cdots \\
\cdots & 0 & -\cos(\beta - \phi_6) l_{\text{leg}} m_{\text{leg}} & 0 & \cdots
\end{bmatrix}
$$

$$
\begin{bmatrix}
\cdots & -\cos(\beta - \phi_4) l_{\text{leg}} m_{\text{leg}} & -\cos(\beta - \phi_5) l_{\text{leg}} m_{\text{leg}} & -\cos(\beta - \phi_6) l_{\text{leg}} m_{\text{leg}} \\
\cdots & -\sin(\beta - \phi_4) l_{\text{leg}} m_{\text{leg}} & -\sin(\beta - \phi_5) l_{\text{leg}} m_{\text{leg}} & -\sin(\beta - \phi_6) l_{\text{leg}} m_{\text{leg}} \\
\cdots & J_l - l_{\text{leg}}^2 m_{\text{leg}} & -m_{\text{leg}} l_{\text{leg}}^2 - \sin(\phi_5) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l & -m_{\text{leg}} l_{\text{leg}}^2 - \sin(\phi_6) m_{\text{leg}} p_{\text{leg}} l_{\text{leg}} + J_l \\
\cdots & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 \\
\cdots & 0 & 0 & 0 \\
\cdots & m_{\text{leg}} l_{\text{leg}}^2 + J_l & 0 & 0 \\
\cdots & 0 & m_{\text{leg}} l_{\text{leg}}^2 + J_l & 0 \\
\cdots & 0 & 0 & m_{\text{leg}} l_{\text{leg}}^2 + J_l
\end{bmatrix}
\tag{A-3}
$$

$$
C(\boldsymbol{q},\dot{\boldsymbol{q}}) = m_{\text{leg}} l_{\text{leg}} \left[\begin{array}{cccccc}
0 & 0 & \sum_{i=1}^{6}\left(\left(-\dot{\beta}+\dot{\phi}_i\right)\sin\left(\beta-\phi_i\right)\right) & -\sin\left(\beta-\phi_1\right)\left(\dot{\phi}_1-\dot{\beta}\right) & -\sin\left(\beta-\phi_2\right)\left(\dot{\phi}_2-\dot{\beta}\right) & \dots \\
0 & 0 & \sum_{i=1}^{6}\left(\left(\dot{\beta}-\dot{\phi}_i\right)\cos\left(\beta-\phi_i\right)\right) & \cos\left(\beta-\phi_1\right)\left(\dot{\phi}_1-\dot{\beta}\right) & \cos\left(\beta-\phi_2\right)\left(\dot{\phi}_2-\dot{\beta}\right) & \dots \\
0 & 0 & p_{\text{leg}}\left(\sum_{i=1}^{2}\left(-\cos\left(\phi_i\right)\dot{\phi}_i\right)+\sum_{i=5}^{6}\left(\cos\left(\phi_i\right)\dot{\phi}_i\right)\right) & \cos\left(\phi_1\right)p_{\text{leg}}\left(\dot{\phi}_1-\dot{\beta}\right) & \cos\left(\phi_2\right)p_{\text{leg}}\left(\dot{\phi}_2-\dot{\beta}\right) & \dots \\
0 & 0 & \cos\left(\phi_1\right)\dot{\beta}p_{\text{leg}} & 0 & 0 & \dots \\
0 & 0 & \cos\left(\phi_2\right)\dot{\beta}p_{\text{leg}} & 0 & 0 & \dots \\
0 & 0 & 0 & 0 & 0 & \dots \\
0 & 0 & -\cos\left(\phi_5\right)\dot{\beta}p_{\text{leg}} & 0 & 0 & \dots \\
0 & 0 & -\cos\left(\phi_6\right)\dot{\beta}p_{\text{leg}} & 0 & 0 & \dots
\end{array}\right.
$$

$$
\left.\begin{array}{cccccc}
\dots & -\sin\left(\beta-\phi_3\right)\left(\dot{\phi}_3-\dot{\beta}\right) & -\sin\left(\beta-\phi_4\right)\left(\dot{\phi}_4-\dot{\beta}\right) & -\sin\left(\beta-\phi_5\right)\left(\dot{\phi}_5-\dot{\beta}\right) & -\sin\left(\beta-\phi_6\right)\left(\dot{\phi}_6-\dot{\beta}\right) \\
\dots & \cos\left(\beta-\phi_3\right)\left(\dot{\phi}_3-\dot{\beta}\right) & \cos\left(\beta-\phi_4\right)\left(\dot{\phi}_4-\dot{\beta}\right) & \cos\left(\beta-\phi_5\right)\left(\dot{\phi}_5-\dot{\beta}\right) & \cos\left(\beta-\phi_6\right)\left(\dot{\phi}_6-\dot{\beta}\right) \\
\dots & 0 & 0 & \cos\left(\phi_5\right)p_{\text{leg}}\left(\dot{\beta}-\dot{\phi}_5\right) & \cos\left(\phi_6\right)p_{\text{leg}}\left(\dot{\beta}-\dot{\phi}_6\right) \\
\dots & 0 & 0 & 0 & 0 \\
\dots & 0 & 0 & 0 & 0 \\
\dots & 0 & 0 & 0 & 0 \\
\dots & 0 & 0 & 0 & 0 \\
\dots & 0 & 0 & 0 & 0 \\
\dots & 0 & 0 & 0 & 0
\end{array}\right] \tag{A-4}
$$

$$
\boldsymbol{J}(\boldsymbol{q}) = \left[\begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & l_{\mathrm{leg}}\cos(\beta-\phi_1) - \sin(\beta)p_{\mathrm{leg}} & l_{\mathrm{leg}}(-\cos(\beta-\phi_1)) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & l_{\mathrm{leg}}\sin(\beta-\phi_1) + \cos(\beta)p_{\mathrm{leg}} & l_{\mathrm{leg}}(-\sin(\beta-\phi_1)) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & l_{\mathrm{leg}}\cos(\beta-\phi_2) - \sin(\beta)p_{\mathrm{leg}} & 0 & l_{\mathrm{leg}}(-\cos(\beta-\phi_2)) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & l_{\mathrm{leg}}\sin(\beta-\phi_2) + \cos(\beta)p_{\mathrm{leg}} & 0 & l_{\mathrm{leg}}(-\sin(\beta-\phi_2)) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & l_{\mathrm{leg}}\cos(\beta-\phi_3) & 0 & 0 & l_{\mathrm{leg}}(-\cos(\beta-\phi_3)) & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & l_{\mathrm{leg}}\sin(\beta-\phi_3) & 0 & 0 & l_{\mathrm{leg}}(-\sin(\beta-\phi_3)) & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & l_{\mathrm{leg}}\cos(\beta-\phi_4) & 0 & 0 & 0 & l_{\mathrm{leg}}(-\cos(\beta-\phi_4)) & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & l_{\mathrm{leg}}\sin(\beta-\phi_4) & 0 & 0 & 0 & l_{\mathrm{leg}}(-\sin(\beta-\phi_4)) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & l_{\mathrm{leg}}\cos(\beta-\phi_5) + \sin(\beta)p_{\mathrm{leg}} & 0 & 0 & 0 & 0 & l_{\mathrm{leg}}(-\cos(\beta-\phi_5)) & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & l_{\mathrm{leg}}\sin(\beta-\phi_5) - \cos(\beta)p_{\mathrm{leg}} & 0 & 0 & 0 & 0 & l_{\mathrm{leg}}(-\sin(\beta-\phi_5)) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & l_{\mathrm{leg}}\cos(\beta-\phi_6) + \sin(\beta)p_{\mathrm{leg}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & l_{\mathrm{leg}}(-\cos(\beta-\phi_6)) & 0 \\
0 & 1 & 0 & l_{\mathrm{leg}}\sin(\beta-\phi_6) - \cos(\beta)p_{\mathrm{leg}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & l_{\mathrm{leg}}(-\sin(\beta-\phi_6)) & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right] \tag{A-5}
$$

Appendix B

Matlab code

In this appendix an overview is given of the most important Matlab functions and files used in the experiments. First, the functions regarding the Switching-Max-Plus-linear model are given and second, the functions regarding the episodic Natural Actor-Critic (eNAC) method are listed.

B-1 Switching Max-Plus-linear model

In this section the functions used for the Max-Plus gait generation method are listed. The functions are provided by dr. G.A. Delgado Lopes, Delft Center for Systems and Control (DCSC), Delft University of Technology (TU Delft), and are only modified for readability. The copyright lies completely with dr. G.A. Delgado Lopes and should be contacted if use is made of this code (modified or unmodified).

MPComputeAStar

```
1  function As = MPComputeAStar(A)
2
3
4  As=MPIdentityMatrix(length(A));
5
6  for n=1:length(A)-1
7      As=MPPlus(MPTimes(As,A),A);
8  end
9
10 As=MPPlus(As,MPIdentityMatrix(length(A)));
```

MPComputeEigenVector

```
function ev = MPComputeEigenVector(gait,Td,Tf,size)

ev = zeros(2*size,1);

for i=1:length(gait)
    for j=1:length(gait{i})
        ev(size+gait{i}{j})=(i-1)*(Tf+Td);
        ev(gait{i}{j})=(i-1)*(Tf+Td)+Tf;
    end
end

ev = ev';
```

MPGenerateAllMatrices

```
function [A,G,H,P,Q] = MPGenerateAllMatrices(gait,number_legs,Tf,Tg,Td)

[P,Q] = MPGeneratePQMatrices(gait,number_legs,Td);

[G,H] = MPGenerateGHMatrices(P,Q,Tf,Tg);

A = MPTimes(MPComputeAStar(G),H);
```

MPGenerateGHMatrices

```
function [G,H]=MPGenerateGHMatrices(P,Q,Tf,Tg)

size=length(P);

G=[MPNullMatrix(size) (Tf+MPIdentityMatrix(size)) ; ...
    P MPNullMatrix(size)];

H=[MPNullMatrix(size) MPNullMatrix(size); ...
    MPPlus(Tg+MPIdentityMatrix(size),Q) MPNullMatrix(size)];
```

MPGeneratePQMatrices

```
function [P,Q] = MPGeneratePQMatrices(gait,number_legs,Td)

P=MPNullMatrix(number_legs);
Q=MPNullMatrix(number_legs);

len = length(gait);

for i=1:len-1
  P([gait{i+1}{:}],[gait{i}{:}]) = Td;
end

Q([gait{1}{:}],[gait{len}{:}]) = Td;
```

MPIdentityMatrix

```
function I = MPIdentityMatrix(size)

I = -Inf*ones(size);

for i=1:size
    I(i,i)=0;
end
```

MPNullMatrix

```
function E = MPNullMatrix(size)

E = -Inf(size);
```

MPPlus

```
function C = MPPlus(A,B)

C=max(A,B);
```

MPTimes

```
function x = MPTimes(a,b)
% MPTIMES
% Revision: vectorized a bit (Fankai Zhang)

x=zeros(size(a,1),size(b,2));
for i=1:size(b,2)
    x(:,i) = max(bsxfun(@plus,a',b(:,i)),[],1)';
end
```

B-2 episodic Natural Actor-Critic

In this section the functions used for the eNAC method are listed. Some of these files are (partially) based on files contained in the Policy Gradient Library, developed by J. Peters, which can be downloaded from http://www.robot-learning.de/Research/PolicyGradientToolbox.

Gradient_eNAC

```
function [w,v,J] = Gradient_eNAC(data,policy,Episode,gamma,...
                                 sigma,J_episode)
% ------------------------------------------------------------------------
% This function calculates the natural gradient, given the input data.
% Inputs:
%   data - data structure containing the states, actions and rewards
%   policy - policy used (structure)
%   Episode - current episode number
%   gamma - discount factor [0,1] (if gamma==1 -> average return)
```

```
10 %   sigma - standard deviation of the normal distribution used in the
11 %           policy
12 %   J_episode - all the returns from the episodes within a single
13 %               parameter update
14 % Ouputs:
15 %   w - natural gradient
16 %   v - value of the start state
17 %   J - return
18 %
19 % Jurriaan Knobel
20 % Master Thesis - Learning optimal gait parameters
21 % using the episodic Natural Actor-Critic method
22 % Delft Center for Systems and Control
23 % Delft Unversity of Technology
24 %
25 % Last modified: 2011/03/15
26 % Copyright: Jurriaan Knobel, 2011
27 %
28 % This code is (partially) based on the Policy Gradient Toolbox
29 % developed by Jan Peters, which can be downloaded from:
30 % http://www.robot-learning.de/Research/PolicyGradientToolbox
31 % ----------------------------------------------------------------------
32
33 % make necessary variables global
34 global M b
35
36 % total return received during episode (average or discounted)
37 J = TotalReturn(data,gamma,0);
38
39 % calculate the average return of all episodes within a single
40 % parameter update, used for the average reward
41 J_episode = [J_episode,J];
42 J_avg     = sum(J_episode)/length(J_episode);
43
44 % derive correct matrix and vector sizes
45 m1 = max(size(GradientLogPolicy(policy,data.state(:,1),...
46       data.action(:,1),sigma)));
47
48 % initialization of necessary matrices and vectors
49 if Episode == 1
50     % initialize matrix
51     M = zeros(m1+1,m1+1);
52     % initialize vector
53     b = zeros(m1+1,1);
54 end
55
56 % initialize Psi
57 Psi = [zeros(m1,1)', 1];
58
59 % initialize total_reward
60 total_reward = 0;
61
62 % loop used to calculate Psi and total_reward
```

```
63 for Step = 1:max(size(data.state(1,:)))
64     % retrieve state
65     s = data.state(:,Step);
66     % retrieve action
67     a = data.action(:,Step);
68 
69     % update Psi
70     Psi = Psi + gamma^(Step-1)*[GradientLogPolicy(policy,s,a,sigma)',0];
71 
72     % update total_reward
73     if gamma == 1 % average return
74         total_reward = total_reward + (data.reward(Step)-J_avg);
75     else % discounted return
76         total_reward = total_reward + gamma^(Step-1)*data.reward(Step);
77     end
78 end
79 
80 % update M-matrix
81 M = M + transpose(Psi)*Psi;
82 
83 % update b-vector
84 b = b + transpose(Psi)*total_reward;
85 
86 % calculate gradient
87 wv = pinv(M)*b;
88 w = wv(1:(max(size(wv)-1)));
89 v = wv(max(size(wv)));
```

TotalReturn

```
1  function J = TotalReturn(data,gamma,J_0)
2  % ------------------------------------------------------------------------
3  % This function calculates the return of the dataset 'data', given the
4  % discount factor gamma and the initial return 'J_0'.
5  %
6  % Inputs:
7  %   data - data set containing the rewards received
8  %   gamma - discount factor (if gamma==1 -> average return)
9  %   J_0 - initial return
10 % Ouputs:
11 %   J - average return
12 %
13 % Jurriaan Knobel
14 % Master Thesis - Learning optimal gait parameters
15 % using the episodic Natural Actor-Critic method
16 % Delft Center for Systems and Control
17 % Delft Unversity of Technology
18 %
19 % Last modified: 2011/02/20
20 % Copyright: Jurriaan Knobel, 2011
21 %
22 % This code is (partially) based on the Policy Gradient Toolbox
23 % developed by Jan Peters, which can be downloaded from:
```

```
% http://www.robot-learning.de/Research/PolicyGradientToolbox
% ----------------------------------------------------------------------

% initial return
J = J_0;

if gamma == 1 % average return
    J = J + sum(data.reward)/max(size(data.reward));
else % discounted return
    for Steps = 1:max(size(data.reward))
        J = J + gamma^(Steps-1) * data.reward(Steps);
    end
end
```

GradientLogPolicy

```
function dlogpi = GradientLogPolicy(policy,s,a,sigma)
% ----------------------------------------------------------------------
% This function calculates the result of the gradient of the logarithm
% of the policy with respect to its parameter vector, at a specific
% state and action. The standard deviations of the normal distribution
% are included the parameter vector at the end (last two values).
%
% Inputs:
%   policy - policy used (structure)
%   s - state values (8*1) at the current time step
%   a - action values (2*1) at the current time step
%   sigma - standard deviation of the normal distribution used in the
%           policy
% Ouput:
%   dlogpi - column vector containing the values of the gradient
%
% Jurriaan Knobel
% Master Thesis - Learning optimal gait parameters
% using the episodic Natural Actor-Critic method
% Delft Center for Systems and Control
% Delft Unversity of Technology
%
% Last modified: 2011/02/24
% Copyright: Jurriaan Knobel, 2011
%
 ----------------------------------------------------------------------

% split gradient up in two parts
dpol1 = policy.gradient(1:policy.theta.sp1);
dpol2 = policy.gradient(policy.theta.sp1+1:policy.theta.sp1+...
             policy.theta.sp2);

% substitute state values in policy
policy.function.policy = subs(policy.function.policy,...
                     policy.state.sym.s,s);

% first action (lift-off angle)
```

```
38 % substitute state values in the gradient
39 dpol1 = subs(dpol1,policy.state.sym.s,s);
40
41 % results of first policy
42 pol1 = policy.function.policy(1);
43
44 % gradient of the logarithm of the policy wrt parameters
45 dlogpi1 = (a(1)-pol1)*dpol1/sigma(1)^2;
46
47 % second action (touchdown angle)
48 % substitute state values in the gradient
49 dpol2 = subs(dpol2,policy.state.sym.s,s);
50
51 % results of second policy
52 pol2 = policy.function.policy(2);
53
54 % gradient of the logarithm of the policy wrt parameters
55 dlogpi2 = (a(2)-pol2)*dpol2/sigma(2)^2;
56
57 % gradient wrt to standard deviation 1
58 dlogsd1 = -1/sigma(1) + (a(1)-pol1)^2/(sigma(1)^3);
59
60 % gradient wrt to standard deviation 1
61 dlogsd2 = -1/sigma(2) + (a(2)-pol2)^2/(sigma(2)^3);
62
63 % total gradient of the logarithm of the policy wrt parameters.
64 dlogpi = [dlogpi1;dlogpi2;dlogsd1;dlogsd2];
65
66 % convert from sym to double
67 dlogpi = double(dlogpi);
```

Bibliography

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Baccelli, F., Cohen, G., Olsder, G., and Quadrat, J. (1992). Synchronization and linearity: an algebra for discrete event systems. *Wiley*.

Baird, L. (1994). Reinforcement learning in continuous time: advantage updating. *Proceedings of the International Conference on Neural Networks*, 4:2448 – 2453 vol.4.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2007). Natural-gradient actor-critic algorithms. *Automatica*.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2009a). Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2009b). Natural actor-critic algorithms, technical report. *Department of Computer Science, University of Alberta, Canada*.

Boyce, W. and DiPrima, R. (2005). Elementary differential equations and boundary value problems. *John Wiley & Sons, New York*, (8).

Buşoniu, L., Babuška, R., De Schutter, B., and Ernst, D. (2010). Reinforcement learning and dynamic programming using function approximators. *CRC Press, Automation and control engineering series*.

Butkovič, P. (2010). Max-linear systems: Theory and algorithms. *Springer Monographs in Mathematics*.

Chernova, S. and Veloso, M. (2004). An evolutionary approach to gait learning for four-legged robots. *Proceedings IROS, IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3:2562–2567.

Cunninghame-Green, R. (1962). Describing industrial processes with interference and approximating their steady-state behaviour. *Operations Research Quarterly*, 13(1):95–100.

Cunninghame-Green, R. (1979). Minimax algebra. *Lecture Notes in Economics and Mathematical Systems*, 166.

Faber, F. and Behnke, S. (2007). Stochastic optimization of bipedal walking using gyro feedback and phase resetting. *Proceedings IEEE-RAS International Conference on Humanoid Robots*, pages 203–209.

Franklin, G., Powell, J., and Emami-Naeini, A. (2006). Feedback control of dynamic systems, 5th edition. *Pearson Prentice Hall*, pages 42–43.

Gabrielli, G. and Von Karman, T. (1950). What price speed? *Mechanical Engineering*, 72(10):775–781.

Giffler, B. (1960). Mathematical solution of production planning and scheduling problems. *IBM Advanced Systems Development Division*.

Greensmith, E., Bartlett, P., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530.

Gregorio, P., Ahmadi, M., and Buehler, M. (1997). Design, control, and energetics of an electrically actuated legged robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 27(4):626–634.

Grondman, I., Vaandrager, M., Buşoniu, L., Babuška, R., and Schuitema, E. (2011). Actor-critic control with reference model learning. *Accepted at The 18th IFAC World Congress*.

Heidergott, B., Olsder, G., and van der Woude, J. (2006). Max plus at work: Modeling and analysis of synchronized systems. *Kluwer*.

Holmes, P., Full, R. J., Koditschek, D., and Guckenheimer, J. (2006). The dynamics of legged locomotion: Models, analyses, and challenges. *SIAM Review*, 48(2):207–304.

Hoyt, D. and Taylor, R. (1981). Gait and the energetics of locomotion in horses. *Nature*, 292(5820):239–240.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642–653.

Kakade, S. (2001). A natural policy gradient. *NIPS*, pages 1531–1538.

Kim, B., Park, J., Park, S., and Kang, S. (2010). Impedance learning for robotic contact tasks using natural actor-critic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(2):433 – 443.

Kim, M. and Uther, W. (2003). Automatic gait optimisation for quadruped robots. *Proceedings of 2003 Australasian conference on robotics and automation*.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. *Proceedings ICRA, IEEE International Conference on Robotics and Automation*, pages 2619–2624.

Konda, V. R. and Tsitsiklis, J. N. (1999). Actor-critic algorithms. *NIPS*, pages 1008–1014.

Kumar, V. and Waldron, K. J. (1988). Force distribution in closed kinematic chains. *IEEE Journal of Robotics and Automation*, 4(6):657 – 664.

Lee, K., Koo, T., and Yoon, Y. (1998). Real-time dynamic simulation of quadruped using modified velocity transformation. *Proceedings 1998 IEEE International Conference on Robotics and Automation*, 2:1701 – 1706 vol.2.

Lopes, G., Babuška, R., De Schutter, B., and van den Boom, T. (2009). Switching max-plus models for legged locomotion. *Proceedings ROBIO, IEEE International Conference on Robotics and Biomimetics*, pages 221–226.

Lopes, G., De Schutter, B., van den Boom, T., and Babuška, R. (2010). Modeling and control of legged locomotion via switching max-plus systems. *Proceedings of the 10th International Workshop on Discrete Event Systems (WODES 2010)*, pages 392–397.

Matsubara, T., Morimoto, J., Nakanishi, J., Sato, M., and Doya, K. (2006). Learning cpg-based biped locomotion with a policy gradient method. *Robotics and Autonomous Systems*, 54:911–920.

Nakamura, Y., Mori, T., Sato, M., and Ishii, S. (2007). Reinforcement learning for a biped robot based on a cpg-actor-critic method. *Neural Networks*, 20(6):723–735.

Nishii, J. (2000). Legged insects select the optimal locomotor pattern based on the energetic cost. *Biological Cybernetics*, 83(5):435–442.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. *Proceedings IROS, IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225.

Peters, J. and Schaal, S. (2007). Policy learning for motor skills. *Proceedings ICONIP, International Conference on Neural Information Processing*, pages 233–242.

Peters, J. and Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Peters, J., Schaal, S., and Schölkopf, B. (2007). Towards machine learning of motor skills. *Proceedings Autonome Mobile Systeme 2007*, pages 138–144.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. *Proceedings IEEE/RSJ International Conference on Humanoid Robots*.

Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural actor-critic. *Proceedings ECML, European Machine Learning Conference*.

Sicilliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2009). Robotics - modelling, planning and control. *Springer*.

Sigaud, O. and Buffet, O. (2010). Markov decision processes in artificial intelligence. *ISTE - Wiley*.

Silva, M., Machado, J., and Lopes, A. (2005). Modelling and simulation of artificial locomotion systems. *Robotica*, 23:595–606.

Spong, M., Hutchinson, S., and Vidyasagar, M. (2006). Robot modeling and control. *Wiley*.

Stewart, J. (2003). Calculus, early transcendentals. *Brooks/Cole*.

Sutton, R. S. and Barto, A. (1998). Reinforcement learning: an introduction. *MIT Press, Cambridge, Massachusetts*.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Morgan and Claypool*.

Taga, G. (1995). A model of the neuro-musculo-skeletal system for human locomotion - 1. emergence of basic gait. *Biological Cybernetics*, 73(2):97–111.

Tedrake, R., Zhang, T., and Seung, H. (2004). Stochastic policy gradient reinforcement learning on a simple 3d biped. *Proceedings IROS, IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3:2849 – 2854 vol.3.

Tong, Y. (1990). The multivariate normal distribution. *Springer-Verlag*.

van den Boom, T. and De Schutter, B. (2004). Modelling and control of discrete event systems using switching max-plus-linear systems. *Proceedings of the 7th International Workshop on Discrete Event Systems (WODES '04)*, pages 115–120.

van den Boom, T. and De Schutter, B. (2006). Modelling and control of discrete event systems using switching max-plus-linear systems. *Control Engineering Practice*, 14(10):1199–1211.

Weingarten, J., Lopes, G., Buehler, M., Groff, R., and Koditschek, D. (2004). Automated gait adaptation for legged robots. *Proceedings ICRA, IEEE International Conference on Robotics and Automation*, 3:2153–2158.

Wolff, K., Sandberg, D., and Wahde, M. (2008). Evolutionary optimization of a bipedal gait in a physical robot. *Proceedings CEC, IEEE Congress on Evolutionary Computation*, pages 440–445.

Glossary

List of acronyms

3mE	Mechanical, Maritime and Materials Engineering
AC	Actor-Critic
CoM	Centre of Mass
CPGs	Central Pattern Generators
DCSC	Delft Center for Systems and Control
eNAC	episodic Natural Actor-Critic
MDP	Markov Decision Process
MPL-DES	Max-Plus-linear discrete event systems
MSE	Mean Squared Error
NAC	Natural Actor-Critic
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
SLIP	Spring-Loaded Inverted Pendulum
TD	Temporal Difference
TU Delft	Delft University of Technology
ZMP	Zero-Moment Point

List of symbols

General

f	A function
$\boldsymbol{\psi}$	A parameter vector
$\mathcal{S}^1$	Circle
$\mathbb{E}^n$	Euclidean space of n dimensions
$\mathbb{T}^n$	Set of n circles
$\mathbb{R}$	Set of real numbers

Robot modelling

C	Coriolis matrix
τ	External forces and torques
$\boldsymbol{q}$	Generalized coordinates, internal states
G	Gravitational matrix
$K_{d,g,x}$	Ground damping x-direction
$K_{d,g,z}$	Ground damping x-direction
$K_{p,g,x}$	Ground stiffness x-direction
$K_{p,g,z}$	Ground stiffness z-direction
z	Height CoM body
M	Inertia matrix
u	Input
$\boldsymbol{x}$	Internal state vector
$\boldsymbol{J}$	Jacobian
K	Kinetic energy
$\mathcal{L}$	Lagrangian
x	Lateral position CoM body
J	Moment of inertia
p^0	Point in coordinate frame 0
P	Potential energy
β	Rotation body
ϕ_i	Rotation leg i
R_1^0	Rotation matrix
$SE(n)$	Special Euclidean Group of order n
$SO(n)$	Special Orthogonal Group of order n
H_1^0	Transformation matrix
d_1^0	Translation

Switching Max-Plus-linear model

λ	Cycle time gait
τ_Δ	Double stance time legs
τ_f	Flight time legs
A_g	Gait matrix
τ_g	Ground time legs
E	Identity matrix $\otimes$-operation
$\{L_j\}$	Leg set j
ϕ_l	Lift-off angle legs
$\oplus$	Max operator
ε	Neutral element $\oplus$-operation
e	Neutral element $\otimes$-operation
φ	Phase Max-Plus gait generation
$\otimes$	Plus operator
$\phi_{\text{ref},i}$	Reference trajectory leg i
$t_i(k)$	Time instant leg i lifts off
$l_i(k)$	Time instant leg i touches down
ϕ_t	Touchdown angle legs
$\mathcal{E}$	Zero matrix $\oplus$-operation

Reinforcement Learning

a_k	Action
$\mathbb{A}$	Action space
A^π	Advantage function
b^π	Baseline
$f^{\boldsymbol{w}}$	Compatible function approximation
$\boldsymbol{w}$	Compatible function approximation parameter vector
μ	Deterministic policy
$\boldsymbol{\nu}$	Deterministic policy parameter vector
γ	Discount factor
e	Episode
$\mathcal{R}_s^a$	Expected value next reward
F	Fisher information matrix
$\nabla_{\boldsymbol{\vartheta}}$	Gradient w.r.t. policy parameter vector
α_a	Learning rate actor
α_c	Learning rate critic
π^*	Optimal policy
u	Parameter update step
$\pi^{\boldsymbol{\vartheta}}$	Parametrized policy
π	Policy

$\boldsymbol{\vartheta}$	Policy parameter vector
$\boldsymbol{M}$	Regression matrix eNAC
$\boldsymbol{b}$	Regression vector eNAC
J_k	Return
r_{k+1}	Reward
ρ	Reward function
σ_{ϕ_l}	Standard deviation policy lift-off angle
σ_{ϕ_t}	Standard deviation policy touchdown angle
s_k	State
Q^π	State-action value function
d^π	State probability distribution
$\mathbb{S}$	State space
X	State transition function
$\mathcal{P}^a_{ss'}$	State transition probability
V^π	State value function
$V^{\boldsymbol{\theta}}$	State value function approximation
$\boldsymbol{\phi}$	State value function approximation basis function
$\boldsymbol{\theta}$	State value function approximation parameter vector