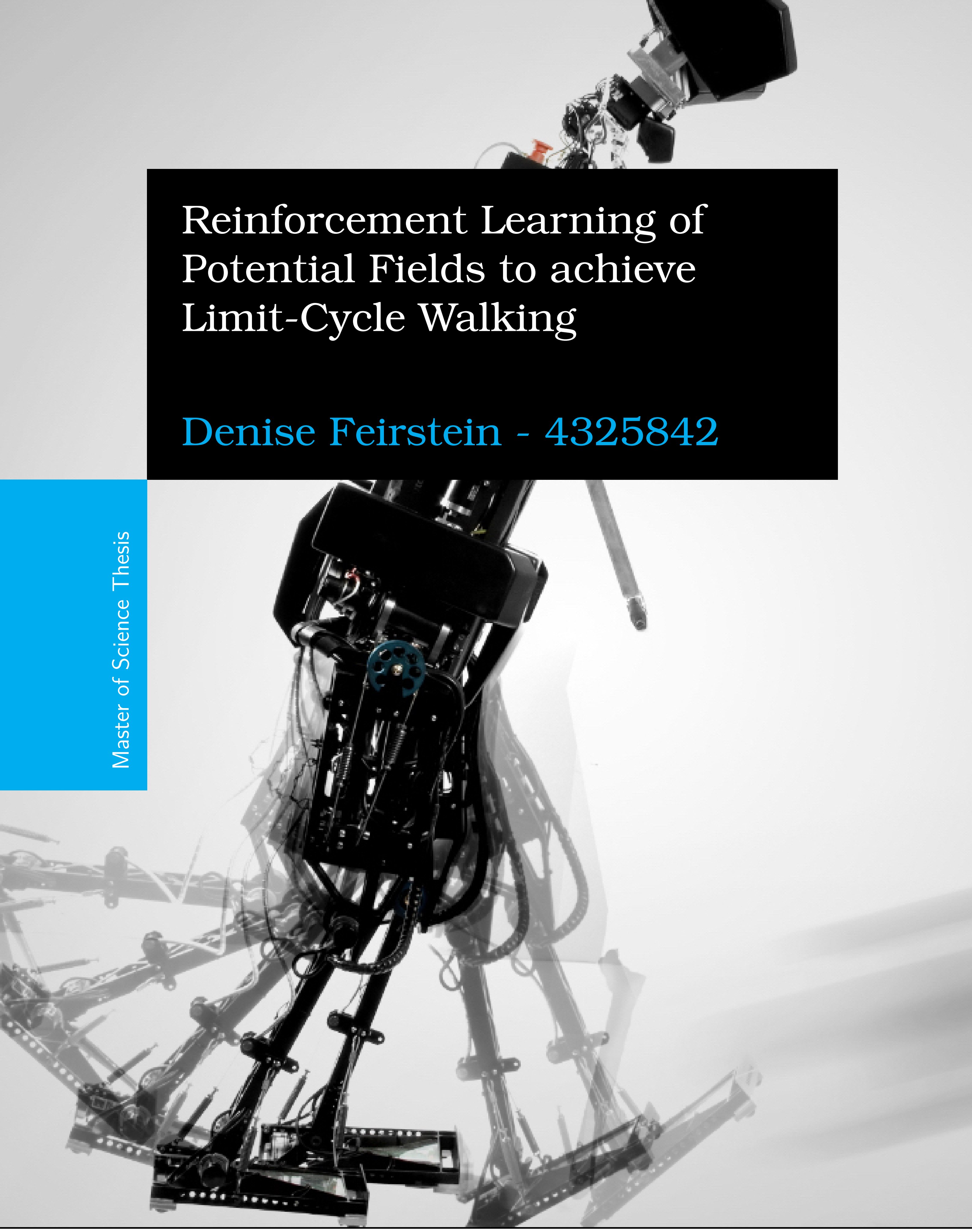

Reinforcement Learning of Potential Fields to achieve Limit-Cycle Walking

Denise Feirstein - 4325842

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Reinforcement Learning of Potential Fields to achieve Limit-Cycle Walking

Master of Science Thesis

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

Denise Feirstein - 4325842

April 4, 2016

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

Delft University of Technology
Department of
BioMechanical Engineering

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

Reinforcement Learning of
Potential Fields to achieve
Limit-Cycle Walking

by

Denise Feirstein - 4325842

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: April 4, 2016

Supervisor(s):	Dr.-Ing. H. Vallery
Reader(s):	Prof.dr.ir. M. Wisse
	Dr.-Ing. J. Kober
	Ir. I. Koryakovskiy

Preface

This thesis, titled "Reinforcement Learning of Potential Fields to achieve Limit-Cycle Walking" is a small piece of a much larger idea that was inspired by Heike Vallery's work "Generalized Elasticities improve patient-cooperative control of rehabilitation robots." I was curious if the virtual potential fields used to control a rehabilitation exoskeleton and guide patients while they relearn to walk could similarly be used to control and guide biped robots. If you have ever worked with TUlip robot, you will understand why I immediately drew a parallel here. This is how my idea of combining potential fields with reinforcement learning was born.

Acknowledgements

This work would not have been possible without the contributions, guidance and support from several people to whom I would like to extend my deepest gratitude.

I would like to thank Heike Vallery for her supervision and guidance. Thank you for sharing in my vision and supporting me throughout this challenging project. My thanks to Jens Kober and Ivan Koryakovskiy for their contribution and guidance in the field of reinforcement learning. Next, I would like to thank Martijn Wisse for helping define the direction and scope of this work. I would also like to thank Joost O. van der Weijde for contributing his knowledge of limit-cycle walking and providing me with his Simplest Walking Model simulation as a foundation for my simulations.

I am grateful to my study group which grew into an "international family" away from home. Thank you, Hulda, Tim, Sarvesh, Marina, Marta (just to name a few) for the much-needed coffee breaks, dinners, conversation, motivation and support.

I am indebted to my parents who have supported me in everything I've done throughout my life. Thank you for instilling in me the work ethic, confidence and sense of adventure that led me to quit my job and pursue something new and unknown.

Finally, I would like to thank my boyfriend for his unwavering support throughout my Masters. Thank you, Heiko, for motivating me to step outside of my comfort zone and always being there for me throughout the challenges.

Abstract

Reinforcement learning is a powerful tool to derive controllers for systems where no models are available. Particularly policy search algorithms are suitable for complex systems, to keep learning time manageable and account for continuous state and action spaces. However, these algorithms demand more insight into the system to choose a suitable controller parameterization. This paper investigates a type of policy parameterization for impedance control that allows energy input to be implicitly bounded: Potential fields. In this work, a methodology for generating a potential field-constrained impedance control via approximation of example trajectories, and subsequently improving the control policy using Reinforcement Learning, is presented. The potential field-constrained approximation is used as a policy parameterization for policy search reinforcement learning and is compared to its unconstrained counterpart. Simulations on a simple biped walking model show the learned controllers are able to surpass the potential field of gravity by generating a stable limit-cycle gait on flat ground for both parameterizations. The potential field-constrained controller provides safety with a known energy bound while performing equally well as the unconstrained policy.

Table of Contents

Chapter 1

Introduction

1-1 Background

In this section, the fields of bipedal robots, potential field control, and reinforcement learning, are introduced and presented based on recent literature.

1-1-1 Bipedal Robots

Bipedal robots that can walk like humans are desirable for a number of applications ranging from art and entertainment [14] to home care and disaster relief. Robots that can move like humans are well suited to aid humans in their homes and workplaces. In cases of emergency when a human environment becomes too dangerous for people to operate in, having robots with two legs that can traverse stairs and obstacles would also be extremely beneficial. Additionally, the study of gait synthesis on bipedal robots is a way of increasing the understanding of human walking dynamics [9]. In the field of medicine, this knowledge is used to advance gait rehabilitation devices and prosthetics. However, the biped robots developed thus far lack the versatility, robustness and energy efficiency that the human gait possesses [5]. One example of an advanced biped robot is Honda's ASIMO. Despite being state of the art, ASIMO, which is representative of joint-angle controlled robots, has a high energy demand [4] and cannot fully extend its knees due to mathematical singularities [16].

In response to the high energy demand of the humanoid robot, passive dynamic walkers have been developed that walk down a shallow slope using only the force of gravity and the robot's natural dynamics [15]. These robots possess an extremely energy efficient gait that is remarkably similar to that of humans. The stable periodic gait of a passive dynamic walker is referred to as a limit-cycle (LC). Bipedal robots based on this concept are often referred to as limit-cycle walkers because they exhibit a stable periodic gait without the need for local stability [12]. The Simplest Walking Model (SWM) developed in [7] has been used as a tool to study this paradigm. Unfortunately, the gait of the passive dynamic walker is extremely sensitive to initial conditions and lacks versatility and robustness [8]. Rendering this gait slope invariant and improving it's disturbance rejection has been the focus of many publications including [10].

1-1-2 Potential Field Control

Energy-based control methods view dynamical systems from the perspective of energy transformation. From this view, a control action can be used to add or remove energy from the system, as well as shape the kinetic or potential energy to the desired form. They are particularly appealing as they provide stability results that hold over large domain unlike those obtained using linear design techniques [3]. These methods are fundamentally based on Lyapunov Stability Theory, which states that if a system is near the minimum of the Lyapunov function, it will stay in the area of this minimum. For the system to asymptotically converge to the minimum point, the system must be dissipative, meaning that energy is dissipated from the system.

A universal property of mechanical systems is passivity which is directly related to Lyapunov methods. A dynamical system with input $\boldsymbol{u}$, output $\boldsymbol{y}$ and state $\boldsymbol{q}$ is said to be passive if there exists a continuously differentiable nonnegative definite scalar function $S(\boldsymbol{q})$, called a storage function such that

$$\dot{S} \leq \boldsymbol{y}^T \boldsymbol{u}. \tag{1-1}$$

The storage function is equivalent to the total energy of the system. Equation 1-1 can be written in integral form as

$$S(\boldsymbol{q}(t_1)) - S(\boldsymbol{q}(t_0)) \leq \int_{t_0}^{t_1} \boldsymbol{y}^T \boldsymbol{u} \mathrm{d}t. \tag{1-2}$$

This means the system satisfies the energy-balance equation

$$\underbrace{S(\boldsymbol{q}(t_1)) - S(\boldsymbol{q}(t_0))}_{\text{stored energy}} = \underbrace{\int_{t_0}^{t_1} \boldsymbol{y}^T \boldsymbol{u} \mathrm{d}t}_{\text{supplied}} - \underbrace{d(t)}_{\text{dissipated}} \tag{1-3}$$

where $d(t)$ is a nonnegative function that captures the dissipation effects. It follows that the energy of the uncontrolled system ($\boldsymbol{u} = 0$) is nonincreasing and the total amount of energy that can be extracted from a passive system is bounded [17].

Potential field control introduces a virtual potential field into the system which can store and supply energy. Variations and extensions of this methods have been applied to the bipedal walking problem in [20], [1], and [2] by emulating an artificial slanted gravity field.

The design of generic potential fields remains challenging, particularly for systems that exhibit modeling uncertainties or that operate in unknown environments. In [6] it is hypothesized that finding a potential field that results in desired trajectories during the design phase of a robot can be used to optimize mechanical structure so that desired trajectories are more “natural” to implement. This is because any virtual potential field (simulated by control torques) could be transformed into a real potential field using mechanical elements such as springs. In the next section, the machine learning technique of reinforcement learning is presented, which could be used to find generic potential fields.

1-1-3 Reinforcement Learning

Reinforcement learning (RL) is a powerful technology to derive controllers for systems where limited or no models are available. RL is a machine learning technique that allows a robot to learn through iterative trial-and-error interactions with its environment. The goal of RL is to find an optimal policy π, a mapping of states to actions, that will maximize the expected return J. The problem commonly consists of an Exploration Strategy, an Evaluation Strategy, and an Update Strategy. The Exploration Strategy varies the policy π to determine the trials that a robot takes to explore its environment. The Evaluation Strategy measures the performance of the policy to determine the expected return J for each trial. Finally, the Update Strategy updates the policy in a way to increase the expected return.

Policy search RL methods, also known as actor-only methods, have been found effective for robotic applications due to their ability to handle higher dimensionality and continuous state and action spaces compared to Value-based RL methods [13]. Furthermore, policy search methods have been effectively implemented on bipedal robots [21]. Policy search methods require a parameterized policy, $\pi_{\boldsymbol{w}}$, where $\boldsymbol{w}$ is a parameter vector that can be perturbed for exploration and updated to improve the policy.

An impedance controller where the control torques are a function of the robot's configuration $\boldsymbol{q}$, as defined in [11], can be formulated as $\boldsymbol{\tau}(\boldsymbol{q})$ using parameterized radial basis functions (RBF) and linear least squares methods. An impedance controller of this form is equivalent to a policy π. The impedance controller can be constrained to a potential field by defining the torques as the transposed Jacobian of a potential function. This reduces the number of policy parameters that need to be explored in a policy search problem.

1-2 Objective

In this section, the research question, methodology and outline of this thesis are presented.

1-2-1 Research Question

In this work, we propose to combine RL and potential field-constrained (PF-constrained) impedance control to achieve LC walking for robots that operate in uncertain conditions because:

- PF-constraint provides safety by bounding the energy
- RL provides controllers for systems with modeling uncertainty.

The question arises, can policy search RL be combined with potential field control to achieve LC walking? While the theoretical advantage of a PF-constrained impedance control, specifically passivity and energy boundedness, are presented in literature, the sub-question arises, are there advantages/limitations of this constraint when it comes to applying RL due to the reduced number of policy parameters?

As a first step towards answering these questions, this work presents a methodology for defining a potential field-constrained (PF-constrained) impedance control via Least Squares (LS) optimization and improving it via reinforcement learning.

1-2-2 Methodology

We define an impedance control as a parameterized mapping of configurations to control torques, which is analogous to a policy in Reinforcement Learning (RL) algorithms. An unconstrained and a PF-constrained impedance controller are initialized via approximation of the SWM trajectory and are compared before and after RL for three cases:

- the **reference case** of the simplest walking model (SWM) on a slope
- the **slope-modified case** of the SWM on flat ground
- the **mass-modified case** of the SWM with modified foot mass on flat ground.

While the SWM can only provide limited insight into the capabilities of the method, the methodology can be extended to more advanced models with higher degrees of freedom and possibly a full-scale humanoid robot like TU Delft's TUlip with series elastic actuation for torque control [9]. A flow chart of the methodology is shown in Figure 1-1 as well as the experimental setup of this work and potential extensions for future work.

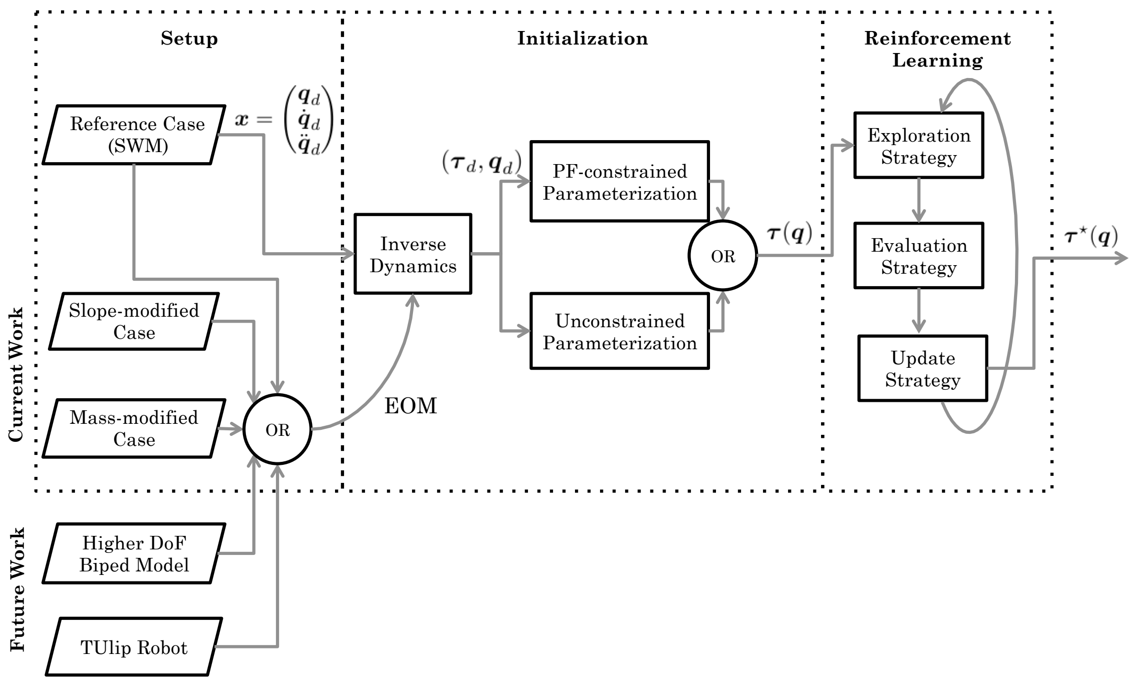

Figure 1-1: Flow chart of methodology for combining Potential Fields (PF) and Reinforcement Learning (RL). The reference trajectory is given by the state vector $x = (q_d\ \dot{q}_d\ \ddot{q}_d)^T$ generated by the ideal SWM on a slope where the subscript d indicates the desired trajectory. For initialization, inverse dynamics with different Equation of Motion (EOM) are used to generate a set of training data (τ_d, q_d) used by LS optimization to derive an impedance control $\tau(q)$, for both the PF-constrained and unconstrained cases. The impedance control is then improved using RL to find an improved control policy $\pi^\star = \tau^\star(q)$.

1-2-3 Outline

The core contribution of this thesis is given in the scientific paper in Chapter 2. A previous version of this paper was accepted to the International Federation of Automatic Control (IFAC) International Workshop on Periodic Control Systems (PSYCO 2016). This paper was the result of the effort of not only myself, but the three other authors. Ivan Koryakovskiy contributed his knowledge of applying reinforcement learning to the bipedal walking problem, specifically with the reward function and exploration algorithm. Jens Kober contributed his expertise of policy search reinforcement learning for robotics, specifically providing the MATLAB code for the PoWER algorithm for reference. Heike Vallery contributed her expertise of bipedal walking and the method of generalized elasticities that she developed, and she provided the MATLAB code for the recursive least squares algorithm as a foundation for this work. All of the authors proofread the paper.

Supplementary material is provided in the Appendix. A formal definition of Lyapunov Stability is given in Appendix A. The equations of motion, impact equations, and energy calculations for the simplest walking model are given in Appendix B. Finally, details of the Least Squares optimization is given in Appendix C, including the recursive algorithm and the radial basis function selection. Selected MATLAB functions are shown in Appendix D to demonstrate the implementation of the methodology.

Chapter 2

Scientific Paper

Reinforcement Learning of Potential Fields to achieve Limit-Cycle Walking

Denise S. Feirstein [*] **Ivan Koryakovskiy** [*] **Jens Kober** [**]
Heike Vallery [*]

[*] *TU Delft Department of BioMechanical Engineering*
[**] *Delft Center for Systems and Control*

Abstract: Reinforcement learning is a powerful tool to derive controllers for systems where no models are available. Particularly policy search algorithms are suitable for complex systems, to keep learning time manageable and account for continuous state and action spaces. However, these algorithms demand more insight into the system to choose a suitable controller parameterization. This paper investigates a type of policy parameterization for impedance control that allows energy input to be implicitly bounded: Potential fields. In this work, a methodology for generating a potential field-constrained impedance control via approximation of example trajectories, and subsequently improving the control policy using Reinforcement Learning, is presented. The potential field-constrained approximation is used as a policy parameterization for policy search reinforcement learning and is compared to its unconstrained counterpart. Simulations on a simple biped walking model show the learned controllers are able to surpass the potential field of gravity by generating a stable limit-cycle gait on flat ground for both parameterizations. The potential field-constrained controller provides safety with a known energy bound while performing equally well as the unconstrained policy.

Keywords: Machine learning, Energy Control, Limit cycles, Walking, Robot control

1. INTRODUCTION

The demand for robot control that is both safe and energy-efficient is greater than ever with advances in mobile robots and robots that interact in human environments. One such example is the bipedal robot which has applications ranging from home care to disaster relief. Traditional position control, common to industrial robotics, is not suitable for robots that interact in unknown environments because slight position errors can result in high contact forces that can damage the robot and its environment. In the case of humanoid robots which interact in human environments this poses a human-safety issue.

One possible solution is to employ impedance control, which attempts to enforce a dynamic relation between system variables as opposed to controlling them directly (Hogan (1984)). Specifically, impedance control based on potential fields, which inherently bounds the energy exchanged between the robot and the environment. Potential fields can modulate natural dynamics of a system and achieve desired behavior without requiring high-stiffness trajectory tracking. Potential fields have been developed for path planning and motion control by reformulating the objective into a potential function (Koditschek (1987)). Control torques can be represented as a vector field generated by the gradient of the potential field, such that the dimensionality of any number of actuators is essentially reduced to one, the scalar value of the potential function.

Potential fields can only release energy stored inside them, such that they can be classified as a passive control method. Motion control based on passivity generates robust motions not only in real time but also autonomously, while allowing simple task objectives, such as walking speed or reaching targets (Hyon and Cheng (2006)). Contrasting the high energy demand of conventional, fully actuated bipedal robots, passive dynamic walkers have been developed that walk down shallow slopes using only the force of gravity and the robot's natural dynamics (McGeer (1990)). Thus, these mechanisms exploit the natural potential field of gravity. In consequence, they possess an extremely energy-efficient gait that is remarkably similar to that of humans. The stable periodic gait of a passive dynamic walker is referred to as a Limit Cycle (LC). Rendering this gait slope-invariant and improving its disturbance rejection has been the focus of many publications including Hobbelen and Wisse (2007). For example, walking of the so-called simplest walker on flat terrain can be achieved by emulating a slanted artificial gravity field via robot actuators (Asano and Yamakita (2001)). This is a very special case of a potential field.

The design and parameterization of more generic potential fields remains challenging, particularly for systems that exhibit modeling uncertainties or are subjected to unknown disturbances. Reinforcement learning (RL) is a powerful technology to derive controllers for systems where no models are available. Policy search RL methods, also known as actor-only methods, have been found effective for robotic applications due to their ability to handle higher dimensionality and continuous state and action spaces compared to Value-based RL methods (Kober et al. (2013)). Furthermore, policy search methods have been effectively implemented on bipedal robots (Tedrake et al. (2004)).

In this work, we propose to combine RL and PF-constrained impedance control to improve robot safety for robots that operate in uncertain conditions because:

- PF-constraint provides safety with a known energy bound
- RL provides controllers for systems with modeling uncertainty.

The question arises, can policy search RL be combined with potential fields to achieve LC walking? While the theoretical advantage of a PF-constrained impedance control, specifically energy boundedness, are presented in literature, the sub-question arises, are there limitations when it comes to RL convergence?

As a first step towards answering these questions, this paper presents a methodology for defining a potential field-constrained (PF-constrained) impedance control and improving it via reinforcement learning. To achieve this, we define an impedance control as a parameterized mapping of configurations to control torques, which is analogous to a policy in Reinforcement Learning (RL) algorithms. A PF-constrained and an unconstrained parameterization of an impedance controller are compared before and after RL applied to the bipedal walking problem. These control methods are compared for three cases: the reference case of the simplest walking model (SWM), the slope-modified case of the SWM on flat ground, and the mass-modified case, of the SWM with modified foot mass on flat ground.

This paper is organized as follows: In Section 2, we describe the parameter optimization method for deriving an initial impedance control policy for an unconstrained and a PF-constrained parameterization. In Section 3, we describe a policy search reinforcement learning algorithm that uses the control policies defined in Section 2 as an initial guess. In Section 4 we describe how this method can be applied to the LC walking problem. In sections 5, we present our evaluation protocol for comparing the unconstrained and PF-constrained impedance control for the bipedal walking. In Section 6, we present our results followed by our discussion in Section 7. Finally, in Section 8 we present our conclusions and suggestions for future work.

2. IMPEDANCE CONTROL INITIALIZATION

As opposed to conventional set-point control approaches that directly control system variables such as position and force, impedance control attempts to enforce a dynamic relation between these variables (Hogan (1984)). In this section, an open-loop impedance controller is derived for a fully actuated robot with n Degrees Of Freedom (DOF) using least squares optimization. The controller is open-loop in that it does not use feedback to determine if the output matches a desired value. We assume an accurate model of the robot as well as the ability to measure the position and torque at each joint as well as full collocated actuation. Each configuration of the robot can be described by a unique vector $\boldsymbol{q} = [q_1, q_2, ..., q_n]^T$ where q_n, with index $i = 1...n$, are the generalized coordinates.

If a desired trajectory, $\boldsymbol{x} = \left(\boldsymbol{q}_\mathrm{d}^T,\ \dot{\boldsymbol{q}}_\mathrm{d}^T,\ \ddot{\boldsymbol{q}}_\mathrm{d}^T\right)^T$, is known, the idealistic control torques, $\boldsymbol{\tau}_0$, required to achieve this trajectory can be found using inverse dynamics. A function to approximate the torques applied to the system as a function of the robot's configuration, $\boldsymbol{\tau}(\boldsymbol{q}) \in \mathbb{R}^n$, can be found by formulating the least squares problem

$$\left(\boldsymbol{\tau}_{0,k}(\boldsymbol{x}_k) - \boldsymbol{\tau}(\boldsymbol{q}_k)\right)^2 \longrightarrow \min \tag{1}$$

where $\boldsymbol{\tau}_{0,k}(\boldsymbol{x}_k)$, $k = 1...S$, is a set of training data with S samples and $\boldsymbol{\tau}(\boldsymbol{q}_k)$ can be approximated as normalized radial basis functions (RBF) $\boldsymbol{G}(\boldsymbol{q})$, parameterized by weighting vector $\boldsymbol{w}$ such that

$$\boldsymbol{\tau}_k = \boldsymbol{G}(\boldsymbol{q}_k)\boldsymbol{w} \tag{2}$$

The choice of $\boldsymbol{G}(\boldsymbol{q})$ will be discussed the the following subsections.

Defining vector $\boldsymbol{b} = (\tau_{0,1}...\tau_{0,S})$ and matrix $\mathbf{A} = \left(\boldsymbol{G}(\boldsymbol{q}_1)...\boldsymbol{G}(\boldsymbol{q}_S)\right)$, the least squares estimate of $\boldsymbol{w}$, denoted $\hat{\boldsymbol{w}}$ can be formulated as the minimization problem

$$\min_{\hat{\boldsymbol{w}}} \|\boldsymbol{b} - \mathbf{A}\hat{\boldsymbol{w}}\|^2_{\mathbf{Q}} \tag{3}$$

which is dependent on the number of training samples, S. The symmetric positive definite weighting matrix $\mathbf{Q}$ contains weights that reflect the importance of certain joints or training samples. The parameter vector $\boldsymbol{w}$ can be found using the pseudoinverse. The solution can also be found recursively if there is a large amount of training data. The procedure for recursive least squares given by Papageorgiou (2012) was modified to include weighting of various parameters such as training data, joints, and torque magnitude.

2.1 Unconstrained Parameterization

The vector function $\boldsymbol{\tau}(\boldsymbol{q})$ can be defined in terms of its components $\tau_i(\boldsymbol{q})$, $i = 1...n$, where n is the number of degrees of freedom, and parameterized as normalized radial basis functions of the form

$$\tau_i(\boldsymbol{q}) = \frac{\sum_{j=1}^{N} w_{i,j} f_j[r_j(\boldsymbol{q})]}{\sum_{j=1}^{N} f_j[r_j(\boldsymbol{q})]} = \boldsymbol{g}(\boldsymbol{q})^T \boldsymbol{w}_i \tag{4}$$

where N is the number of basis functions, $w_{i,j}$ is the j^{th} parameter of the i^{th} weighting vector, f_j is an RBF, r_j is a radius function and $\boldsymbol{g}(\boldsymbol{q})$ is a vector function. For the unconstrained case $\boldsymbol{g}(\boldsymbol{q})$ is used as $\boldsymbol{G}(\boldsymbol{q})$ in Equation 2.

Radius functions r_i are scalar functions of the distance vector $\boldsymbol{\delta}_i$ and scaling factor s which defines the size of the radial basis function:

$$r_j(\boldsymbol{q}) = s\,\|\boldsymbol{\delta}_j\|\,. \tag{5}$$

$\boldsymbol{\delta}_j$ describes the distance from the center point $\boldsymbol{c}_j$ of the j^{th} RBF to the joint configurations $\boldsymbol{q}$:

$$\boldsymbol{\delta}_j(\boldsymbol{q}) = \boldsymbol{q} - \boldsymbol{c}_j. \tag{6}$$

For the RBF, f_j, we choose to use compactly supported radial basis functions which allow for the use of a minimal number of center points c_j in the neighborhood of the robot's position to sufficiently compute the function value (Vallery et al. (2009a)). This reduces the computational resources needed during operation.

2.2 Potential Field-constrained Parameterization

Function $\boldsymbol{\tau}(\boldsymbol{q})$ can be constrained to describe a potential field by enforcing that its work is zero for any closed-path trajectory. This implies the control torques are a function of the joint variables $\boldsymbol{q}$ and can be defined as the negative gradient of a potential function $\psi(\boldsymbol{q})$ with respect to $\boldsymbol{q}$:

$$\boldsymbol{\tau}(\boldsymbol{q}) = -\nabla_q \psi(\boldsymbol{q}). \tag{7}$$

This is similar to the method of Generalized Elasticities presented in Vallery et al. (2009a) and Vallery et al. (2009b).

Similar to Equation 4, potential function $\psi(\boldsymbol{q})$ can be parameterized as normalized radial basis functions (RBF) of the form

$$\psi(\boldsymbol{q}) = \frac{\sum_{j=1}^{N} w_j f_j[r_j(\boldsymbol{q})]}{\sum_{j=1}^{N} f_j[r_j(\boldsymbol{q})]} = \boldsymbol{g}(\boldsymbol{q})^T \boldsymbol{w}. \tag{8}$$

Unlike the unconstrained parameterization, which requires a unique weighting vector $\boldsymbol{w}_i$ for each degree of freedom, for the PF-constrained parameterization, the torques can be formulated as the gradient of the potential shown in Equation (7). This can be estimated as the transposed Jacobian of $\boldsymbol{g}(\boldsymbol{q})$:

$$\boldsymbol{\tau}(\boldsymbol{q}) = -\left(\frac{\partial \boldsymbol{g}(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)^T \boldsymbol{w}. \tag{9}$$

where $\boldsymbol{G}(\boldsymbol{q}) = -\left(\frac{\partial \boldsymbol{g}(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)^T$.

3. POLICY SEARCH REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a machine learning method which attempts to find a control policy, $\pi(\boldsymbol{u}|\boldsymbol{x}, \boldsymbol{w})$, which maps states $\boldsymbol{x}$ to actions $\boldsymbol{u}$. For policy search algorithms, the policy is parameterized by a weighting vector $\boldsymbol{w}$. The policy is analogous to the impedance control laws derived in the previous section where generalized coordinates $\boldsymbol{q}$ are states and control torques $\boldsymbol{\tau}$ are actions.

3.1 Exploration Strategy

The policy space is explored by randomly perturbing the weighting vector $\boldsymbol{w}$. Batch exploration is performed where the policy is independently perturbed from the initial policy a set number of times. The perturbed policies are then evaluated and updated according to the strategies in the following sections.

3.2 Evaluation Strategy

The performance of the policy is numerically evaluated by computing the expected return J, which is a sum of the expected reward R. Based on the expected return J the policy is updated with the objective to find a policy which maximizes the expected return J. The policy evaluation strategy determines how to evaluate the performance of an executed policy by using a reward function, $R(\boldsymbol{x}, \boldsymbol{u})$. For a finite-horizon model, this corresponds to maximizing the expected reward for the horizon H over h steps. The series of states and actions over H steps is called an episode. The expected return is calculated

$$J = E\left\{\sum_{h=0}^{H} R_h\right\}. \tag{10}$$

Episode-based policy evaluation uses the entire episode to assess the quality of the policy used directly (Deisenroth et al. (2011)). The episode-based evaluation strategy is summarized in Algorithm 1.

Algorithm 1 Episode-Based Evaluation for finite-horizon model

```
Generate episode using policy $\pi$
for $h = 1$ to $H$ do
    Evaluate: Assess step $h$ assigning reward $R_h$
end for
Compute return $J = \sum_{h=0}^{H} R_h$
```

3.3 Update Strategy

The policy is updated based on the performance of the previous policy or set of policies. Policy search methods optimize around an initial policy $\pi(\boldsymbol{u}|\boldsymbol{x}, \boldsymbol{w}_0)$. The policy is iteratively updated using an update strategy that computes changes in the policy parameter in a way that increases the expected return.

Several update strategies for episode-based policy search have been developed. One method developed in Kober and Peters (2011) specifically for motor primitives in robotics is Expectation Maximization Policy learning by Weighted Exploration with the Returns (PoWER). The iterative policy search method with episode-based evaluation is summarized in Algorithm 2.

Algorithm 2 Policy Search using Expectation Maximization PoWER

```
Initialize: Generate initial episode using policy $\pi_0$ parameterized by $\boldsymbol{w}_0$. Compute return $J_0$.
repeat
    Explore: Perform $i = 1 : N$ episodes using perturbed policy parameters $\boldsymbol{w}_i = \boldsymbol{w}_{i-1} + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$
    For each episode compute return $J_i$
    Reweight: Compute importance weights, keep 10 high-importance episodes, discard low-importance episodes.
    Update: Compute updated policy
    $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \left\langle \sum_i^{10} J_i \right\rangle^{-1} \left\langle \sum_i^{10} \epsilon_i J_i \right\rangle$
until Policy converge
```

4. APPLICATION TO LC WALKING

4.1 Simplest Walking Model

The simplest walking model (SWM) developed in Garcia et al. (1998) is often used as a tool to study the paradigm of Bipedal Limit-Cycle walking and is detailed in the following sections. A diagram of the SWM is shown in Figure 1.

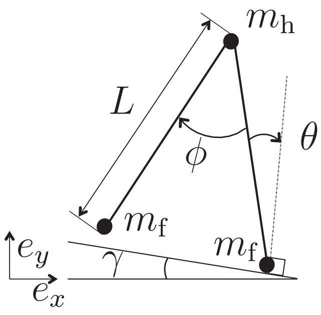

Fig. 1. Diagram of the Simplest Walking Model which consists of two massless links, point mass m_h at the hip and m_f at each foot walking on ground of slope γ. The generalized coordinates θ and ϕ are the angle of the stance leg perpendicular to the slope and the inter-leg (or hip) angle respectively.

The model consists of two massless rigid links of length L connected at the hip by a frictionless hinge. The mass is distributed over three point masses at the hip and feet such that the hip mass m_h is much larger than the foot mass m_f. The model is situated on a slope of angle γ and acts only under the force of gravity with acceleration constant g. The configuration of the model is given by the ankle angle θ and hip angle ϕ. The generalized coordinates are $\boldsymbol{q} = (x_\mathrm{c}, y_\mathrm{c}, \theta, \phi)^T$ where the subscripts "c" denotes the contact point of the stance foot with the ground. The model is actuated at the ankle and hip.

4.2 Inverse dynamics

A vector of the global coordinates of the point masses is $\boldsymbol{p} = (x_\mathrm{st}, y_\mathrm{st}, x_\mathrm{hip}, y_\mathrm{hip}, x_\mathrm{sw}, y_\mathrm{sw})^T$ where subscripts "st" and "sw" denote the stance leg and swing leg respectively and subscript "hip" denotes the hip. The generalized coordinates can be transformed to Cartesian positions using transfer function $\boldsymbol{p} = \boldsymbol{F}(\boldsymbol{q})$. The equations of motion can then be found using the virtual power equation

$$\delta\dot{\boldsymbol{p}}^T[\boldsymbol{f} - \mathbf{M}\ddot{\boldsymbol{p}}] = 0. \tag{11}$$

where $\mathbf{M}$ is the global mass matrix defined $\mathbf{M}$ =Diag$(m_\mathrm{f}, m_\mathrm{f}, m_\mathrm{h}, m_\mathrm{h}, m_\mathrm{f}, m_\mathrm{f})$. The resulting equations of motion are

$$[\mathbf{F}_{,\boldsymbol{q}}^T\mathbf{M}\mathbf{F}_{,\boldsymbol{q}}]\ddot{\boldsymbol{q}} = \mathbf{F}_{,\boldsymbol{q}}^T[\boldsymbol{f}_g - \mathbf{M}\mathbf{F}_{,\boldsymbol{q}\boldsymbol{q}}\dot{\boldsymbol{q}}\dot{\boldsymbol{q}}] + \boldsymbol{Q} \tag{12}$$

where the subscript comma operator followed by $\boldsymbol{q}$ denotes partial derivative by $\boldsymbol{q}$, and $\boldsymbol{f}_g$ are the applied forces due to gravity given

$$\boldsymbol{f}_g = \mathbf{M}[\sin\gamma, -\cos\gamma, \sin\gamma, -\cos\gamma, \sin\gamma, -\cos\gamma]^T \tag{13}$$

and $\boldsymbol{Q} = (Q_{x_c}, Q_{y_c}, Q_\theta, Q_\phi)^T$ are the generalized forces. For unactuated cases, Q_θ and Q_ϕ both equal zero. The contact forces at the stance foot Q_{x_c} and Q_{y_c} are only valid for $Q_{y_c} > 0$. In this case $\boldsymbol{Q}$ is known and $\ddot{\boldsymbol{q}}$ can be found using the ordinary differential equation

$$\ddot{\boldsymbol{q}} = \frac{\mathbf{F}_{,\boldsymbol{q}}^T[\boldsymbol{f}_g - \mathbf{M}\mathbf{F}_{,\boldsymbol{q}\boldsymbol{q}}\dot{\boldsymbol{q}}\dot{\boldsymbol{q}}] + \boldsymbol{Q}}{[\mathbf{F}_{,\boldsymbol{q}}^T\mathbf{M}\mathbf{F}_{,\boldsymbol{q}}]}. \tag{14}$$

The unactuated model exhibits an LC gait for a limited set of combinations of initial conditions and slopes, called the basin of attraction. In the case we would like to deviate from the original basin of attraction (for example by modifying the slope and/or model mass) while still maintaining an LC gait, idealistic actuator torques can be derived using inverse dynamics of the known LC joint trajectories to find the generalized forces. Rearranging Equation 12 and replacing $\mathbf{M}$ and $\boldsymbol{f}_g$ with $\mathbf{M}_\mathrm{mod}$ and $\boldsymbol{f}_{g,\mathrm{mod}}$ respectively, gives the inverse dynamics equation

$$\begin{aligned}\boldsymbol{Q}_0 =& [\mathbf{F}_{,\boldsymbol{q}}^T\mathbf{M}_\mathrm{mod}\mathbf{F}_{,\boldsymbol{q}}]\ddot{\boldsymbol{q}}_d \\ &- \mathbf{F}_{,\boldsymbol{q}}^T[\boldsymbol{f}_{g,\mathrm{mod}} - \mathbf{M}_\mathrm{mod}\mathbf{F}_{,\boldsymbol{q}\boldsymbol{q}}\dot{\boldsymbol{q}}_d\dot{\boldsymbol{q}}_d]\end{aligned} \tag{15}$$

where training data $(\dot{\boldsymbol{q}}_d, \ddot{\boldsymbol{q}}_d)$ is required, $\boldsymbol{f}_{g,\mathrm{mod}}$ corresponds to the applied forces from the modified slope and $\mathbf{M}_\mathrm{mod}$ corresponds to the modified global mass matrix. From this point forward $\boldsymbol{\tau}_0 = [Q_{\theta,0}, Q_{\phi,0}]^T$ will be used to denote the generalized forces at the ankle and hip joints corresponding to the idealistic applied motor torques.

The training data was found by first, scanning the initial conditions $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ for cases in which the SWM converges to an LC and then the associated accelerations $\ddot{\boldsymbol{q}}$ were found using Equation 14. The resulting training data can represented by the vector $\boldsymbol{x} = \left(\boldsymbol{q}_d^T,\ \dot{\boldsymbol{q}}_d^T,\ \ddot{\boldsymbol{q}}_d^T\right)^T = \left(\theta,\ \phi,\ \dot{\theta},\ \dot{\phi},\ \ddot{\theta},\ \ddot{\phi}\right)^T$.

For scanning the initial conditions, the ankle angle was varied between 0.1 and 0.2 rad with a step size of 0.005 rad, and the initial hip angle was set to twice that of the ankle so the model initializes in double support phase. The initial ankle angular velocity was varied between -0.68 and -0.38 rad/s with a step size of 0.005 rad/s, and the initial hip angular velocity was set to 0 rad/s. The result is shown in Figure 2.

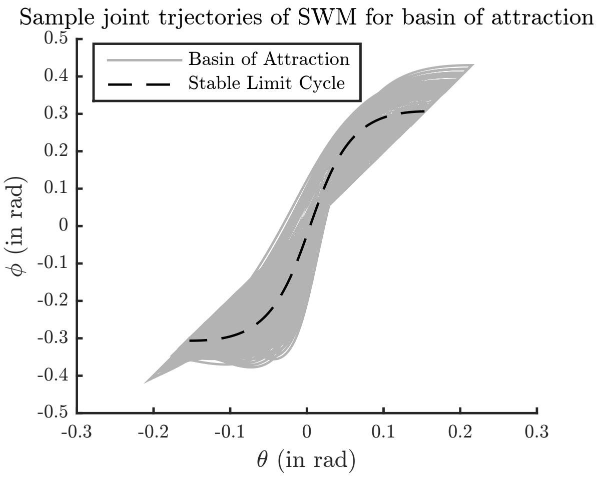

Fig. 2. Joint trajectories of the basin of attraction for the simple walking model with $\gamma = 0.004$ rad, $m_h = 1$ kg, $m_f = 0.001$ kg, $L = 1$ m, and $g = 10$ m/s^2

The torques $\boldsymbol{\tau}_0$ found from training data $\boldsymbol{x}$ can be used to solve the least-squares problem in Equation (3) using the recursive least-squares method described in Section 2 resulting in impedance control laws of the form $\boldsymbol{\tau}(\boldsymbol{q})$.

4.3 Reinforcement Learning

The resulting impedance control laws $\boldsymbol{\tau}(\boldsymbol{q})$ parameterized by vector $\boldsymbol{w}$ are specific to the simplest walking model case and will likely not be effective if the model is modified or more degrees of freedom are added. If this is the case $\boldsymbol{\tau}(\boldsymbol{q})$ parameterized by vector $\boldsymbol{w}_0$ can be used as the initial policy for policy search RL. The policy search with episode-based evaluation strategy described in Section 3 can be used where one episode is H steps of the biped. For a biped robot, the state transitions from the previous state $\boldsymbol{x}$ to the next state $\boldsymbol{x}'$ caused by actions $\boldsymbol{u}$ can be modeled by solving the equations of motion (14) using iterative methods where $\boldsymbol{x} = (\boldsymbol{q}^T,\ \dot{\boldsymbol{q}}^T)^T$ are the states and the generalized forces Q_θ, Q_ϕ are the actions $\boldsymbol{u}$.

To evaluate the quality of parameter vector $\boldsymbol{w}_k$, the return for each episode is calculated

$$J_k = \sum_{h=0}^{H} R_h. \tag{16}$$

The reward function used for each step is

$$\begin{aligned} R_h(\boldsymbol{x}, \boldsymbol{u}) = &+ R_{\text{step}} - R_\Delta\Big(||\Delta\theta|| + ||\Delta\dot{\theta}||\Big) \\ &- R_t||t_h - t_0|| - R_\tau||\boldsymbol{\tau}|| \end{aligned} \tag{17}$$

where $\Delta\theta = \theta_h - \theta_{h-1}$, and $\Delta\dot{\theta} = \dot{\theta}_h - \dot{\theta}_{h-1}$, and R_{step}, R_Δ, R_t and R_τ are constants. The first term of the reward function is given as a reward for successfully completing a step. The second term penalizes the change in angle and angular velocity of the stance leg at the beginning of each step. This is to encourage a limit-cycle is reached where each step is the same. The third term penalized the change in time of step h from the time of the reference LC step t_0. The fourth term penalizes the magnitude of the control torques to minimize the energy added to the system.

5. EVALUATION PROTOCOL

5.1 Implementation

Simulations were performed in MATLAB to assess the impedance controllers described in the previous section. The simulations used the ODE45 integration algorithm with the following settings: absolute tolerance = 10^{-6}, relative tolerance = 10^{-3} and initial integration-step size $\Delta t = 0.02$ s. In the case of an "odezero (internal error)" the ODE45 settings were temporarily set to: absolute tolerance = 10^{-8}, relative tolerance = 10^{-5}. The event detection was used to determine when a step occurs using the step condition: $\theta - \phi/2 = 0$ and $\dot{\theta} < 0$.

The impedance control laws were implemented on a fully-actuated simple walking model for the three cases: the reference case of the simplest walking model (SWM) on a slope, the slope-modified case of the SWM on flat ground, and the mass-modified case, of the SWM with modified foot mass on flat ground. For all cases the leg length, hip mass and gravity remained constant at $L = 1$ m, $m_\text{h} = 1$ kg and $g = 10$ m/s^2 respectively.

For the least squares optimization, 50 RBFs were used. The center locations were determined using a grid step size of 0.05 rad for the ankle angle and 0.1 rad for the hip angle in the area of the ideal trajectory of the SWM as shown in Figure 3.

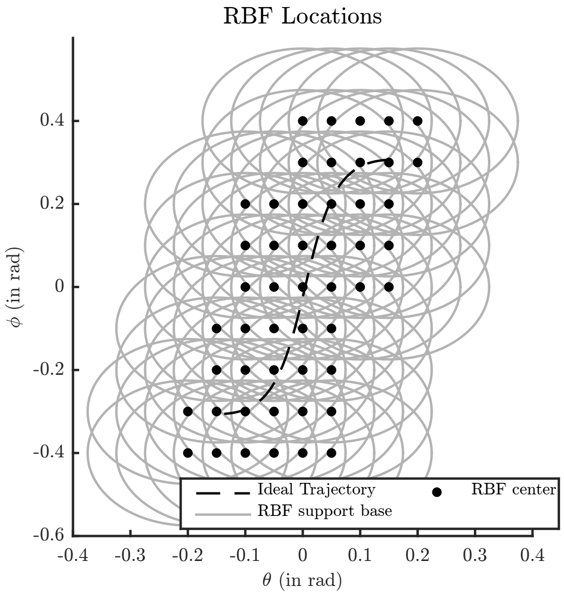

Fig. 3. RBF center locations and support base of 10 degrees.

For the policy search RL, a horizon of $H = 10$ was used corresponding to 10 steps of the robot. For the exploration strategy, a batch size of 100 iterations was used. For the reward function the constants were set to: $R_{\text{step}} = 1$, $R_\Delta = 10$, $R_t = 1$ and $R_\tau = 5$. The time of the reference LC step was $t_0 = 1.2180$ s. A Gaussian exploration $\epsilon \sim \mathcal{N}(0, \sigma^2)$ was used which was decrease linearly over episodes.

5.2 Experiment Setup

Initial unconstrained and PF-constrained impedance controllers were found using inverse dynamics for each of the three cases described below:

Reference case: For the reference case a slope of $\gamma = 0.004$ rad and foot mass $m_\text{f} = 0.001$ kg was used.

Slope-modified case: For the slope-modified case a slope of $\gamma = 0$ rad and foot mass $m_\text{f} = 0.001$ kg was used.

Mass-modified case: For the modified-mass case, a foot mass of $m_\text{f} = 0.01$ kg was used. This is 10 times the value of the reference case. A slope of $\gamma = 0$ rad was used.

For the Slope and Mass-modified cases, RL was used to attempt to improve the policy for both the unconstrained and the PF-constrained parameterizations. For the reference case, the performance of the controllers can not be improved further using RL based on the evaluation strategy since the control torques cannot decrease further.

5.3 Benchmarking Criteria

The unconstrained and PF-constrained impedance controllers were compared for each of the three cases based on the following benchmarking criteria:

Work and Energy: The energy of the LC of the ideal SWM (unactuated and on a slope) is bounded by the potential field of gravity. The energy bound can be measured as the maximum energy, E of the LC, defined $E = V + T$ where V is the potential energy and T is the kinetic energy. For the LC of the ideal SWM, the total energy is constant at 10.0108 J. At each step kinetic energy is dissipated at impact and an equivalent amount of potential energy is added by the slope. The change in kinetic energy at the end of the step can be seen in Figure 4. The energy added/dissipated at each step is equivalent to 0.0166 J.

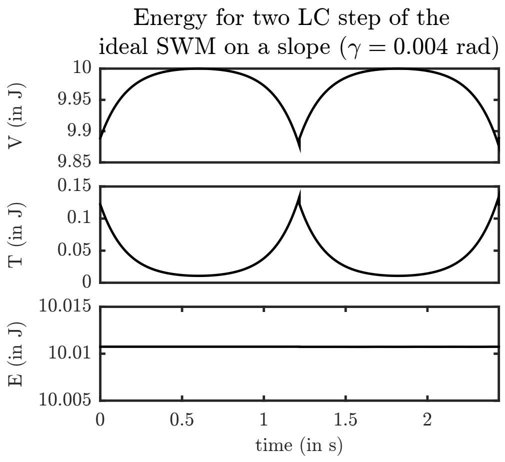

Fig. 4. Energy of the SWM on a slope of $\gamma = 0.004$ rad. The total energy is the sum of the potential (V) and kinetic (T) energy.

Energy consumption can be measured for the actuated model as the work done by the actuators:

$$W = \int_{\boldsymbol{q}_0}^{\boldsymbol{q}_1} \boldsymbol{\tau} \mathrm{d}\boldsymbol{\theta} \tag{18}$$

where $\boldsymbol{q}_0$ is the configuration at the beginning of the step and $\boldsymbol{q}_1$ is the configuration at the end of the step.

Robustness: The robustness of an LC gait can be measured by its velocity disturbance rejection. An angular velocity disturbance is introduced to the stance leg at the beginning of the first step and the maximum disturbance that can be applied without causing the walker to fall is used as a measure for robustness.

RL Performance: The performance of the RL is assessed by plotting the mean performance over the episodes, for several trials, and observing how many episodes it takes to level off.

6. RESULTS

6.1 Reference case

The trajectory phase plots for the Unconstrained and PF-constrained policies derived using inverse dynamics for the reference case are shown in Figures 5 (a) and (b) respectively. The control torque and total energy for the Unconstrained and PF-constrained policy derived using inverse dynamics for the reference case shown in Figure 6 (a) and (b) respectively.

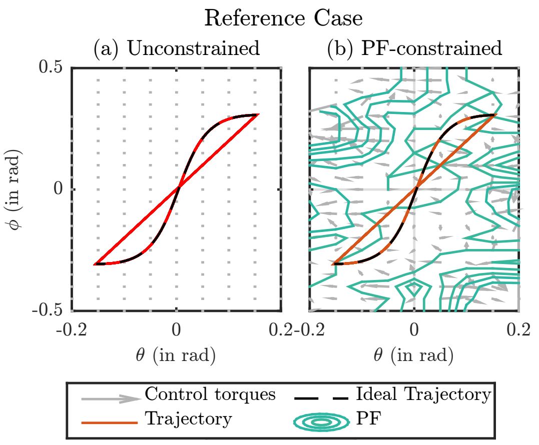

Fig. 5. Trajectory phase plot of the (a) Unconstrained and (b) PF-constrained policies for the Reference Case. The control policies are represented by a vector field and for the PF-constrained policy the contour lines of the potential field are shown.

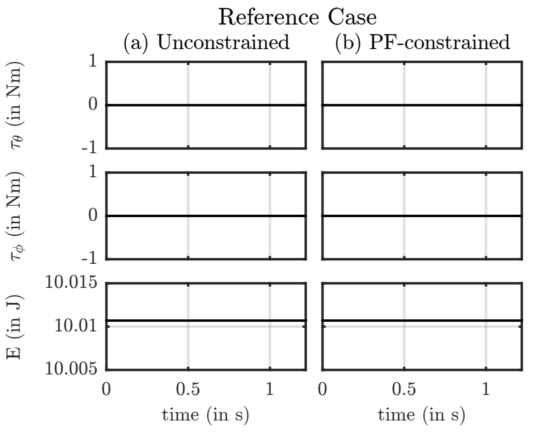

Fig. 6. Control torques and energy of one LC step of the (a) Unconstrained and (b) PF-constrained policies for the Reference Case.

The benchmarking criteria for the energy, work and robustness of the reference case are specified Table 1.

6.2 Slope-modified Case

Initialization The trajectory phase plots for the initial Unconstrained and PF-constrained policies for the Slope-modified case are shown in Figures 7 (a) and (b) respectively. The control torques and total energy for the initial Unconstrained and PF-constrained policies for the Slope-modified case are shown in Figures 8 (a) and (b) respectively.

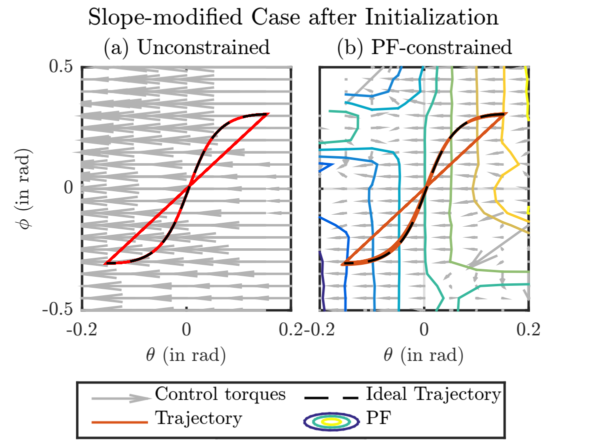

Fig. 7. Trajectory phase plot of the initial (a) Unconstrained and (b) PF-constrained policies for the Slope-modified Case. The control policies are represented by a vector field and for the PF-constrained policy the contour lines of the potential field are shown.

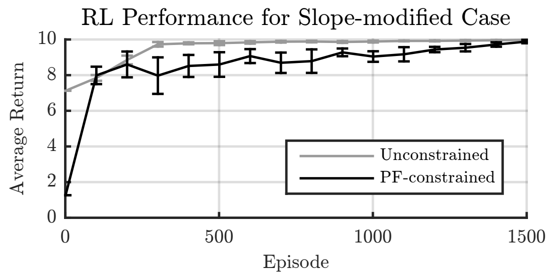

Fig. 9. Mean performance of the RL for the Unconstrained and PF-constrained policies for the Slope-modified case averaged over 10 runs with the error bars indicating the standard deviation. For both policies the exploration variance decreased linearly from 1e-6 to 1e-11 throughout the episodes.

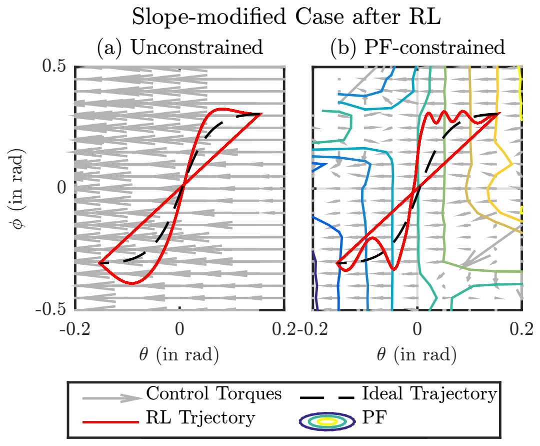

Fig. 10. Trajectory phase plot of the learned (a) Unconstrained and (b) PF-constrained policies for the Slope-modified Case. The control policies are represented by a vector field and for the PF-constrained policy the contour lines of the potential field are shown.

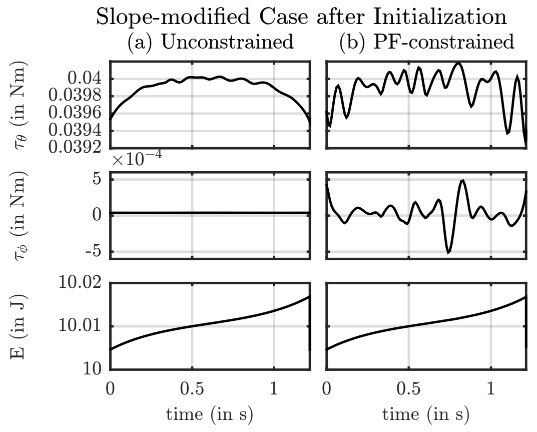

Fig. 8. Control torques and energy of one LC step of the initial (a) Unconstrained and (b) PF-constrained policies for the Slope-modified Case.

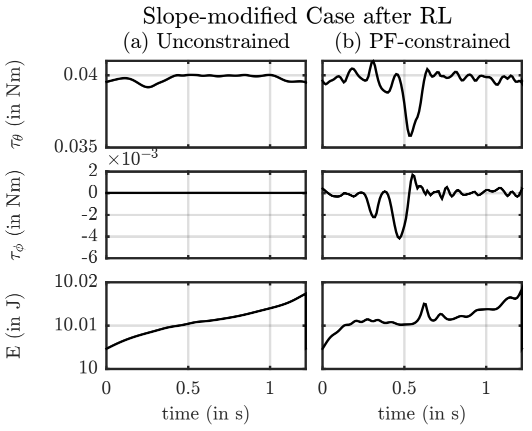

Fig. 11. Control torques and energy of one LC step of the learned (a) Unconstrained and (b) PF-constrained policies for the Slope-modified Case

Reinforcement Learning Results The mean performance of the RL for the Unconstrained and PF-constrained controllers are shown in Figure 9. The resulting trajectory phase plot for the learned Unconstrained and PF-constrained policies for the Slope-modified case are shown in Figures 10 (a) and (b) respectively. The resulting control torques and energy for the learned Unconstrained and PF-constrained policies for the Slope-modified case are shown in Figures 11 (a) and (b) respectively.

The benchmarking criteria for the energy, work and robustness of the Slope-modified case are specified Table 1.

6.3 Mass-modified Case

Initialization The trajectory phase plot for the initial Unconstrained and PF-constrained policies for the Mass-modified case is shown in Figure 12 (a) and (b) respectively.

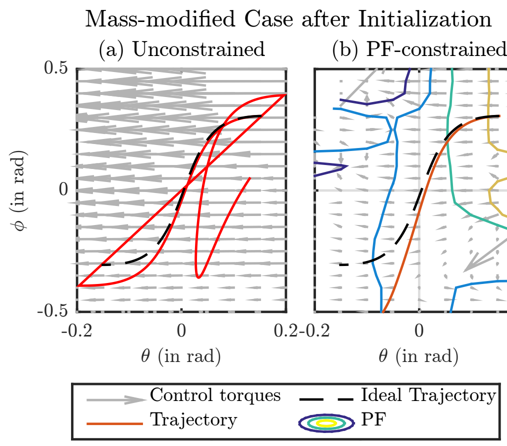

Fig. 12. Trajectory phase plot of the initial (a) Unconstrained and (b) PF-constrained policies for the Mass-modified Case.The control policies are represented by a vector field and for the PF-constrained policy the contour lines of the potential field are shown.

It can be seen that neither policy leads to a stable limit cycle so the corresponding control torque and energy plots are not shown.

Reinforcement Learning The mean performance of the RL for both the PF-constrained and unconstrained case are shown in Figure 13. The resulting trajectory phase plots for the learned Unconstrained and PF-constrained policies for the Mass-modified case are shown in Figures 14 (a) and (b) respectively. The resulting control torques and energy for the learned Unconstrained and PF-constrained policies for the Mass-modified case are shown in Figures 15 (a) and (b) respectively.

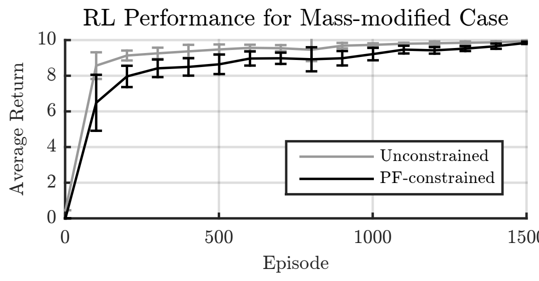

Fig. 13. RL mean performance of the Unconstrained and PF-constrained policies for the Mass-modified case averaged over 10 runs with the error bars indicating the standard deviation. For the Unconstrained policy the exploration variance decreased from 1e-5 to 1e-10 and for the PF-constrained policy the variance decreased from 1e-6 to 1e-10.

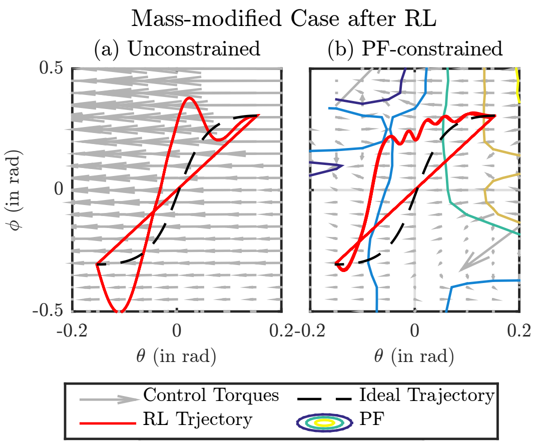

Fig. 14. Trajectory phase plot of the learned (a) Unconstrained and (b) PF-constrained policies for the Mass-modified Case. The control policies are represented by a vector field and for the PF-constrained policy the contour lines of the potential field are shown.

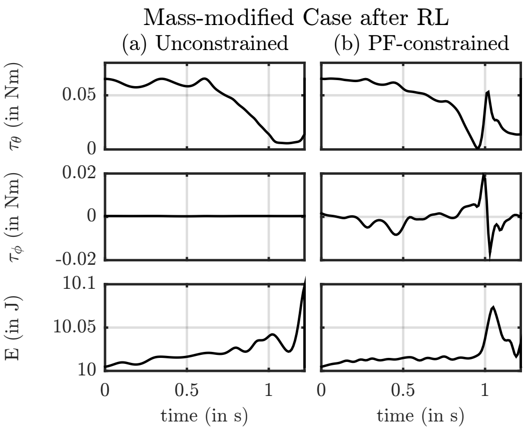

Fig. 15. Control torques and energy of one LC step of the learned (a) Unconstrained and (b) PF-constrained policies for the Mass-modified Case.

The benchmarking criteria for the work, energy and robustness of the Mass-modified case are specified Table 1.

6.4 Results Summary

The benchmarking criteria for the energy bound, work and robustness for each case are summarized in Table 1. The energy bound was determined by the max energy displayed in Figures 6, 8, 11 and 15. The work was calculated using Equation 18. The robustness is given by the max velocity disturbance rejection as described in Section 5.3. The ✗ indicates that an LC was not achieved so there was no benchmarking criteria.

Table 1. Summary of Results

Case	Parameter-ization	Benchmarking Criteria	Initial Policy	Learned Policy
Reference	Unconstrained	Energy bound (J)	10.0107	-
		Work (J)	0	-
		Max velocity disturbance (m/s)	-0.05	-
	PF-constrained	Energy bound (J)	10.0107	-
		Work (J)	5e-14	-
		Max velocity disturbance (m/s)	-0.05	-
Slope-modified	Unconstrained	Energy bound (J)	10.0169	10.0258
		Work (J)	1.5074	1.4877
		Max velocity disturbance (m/s)	-0.05	-0.03
	PF-constrained	Energy bound (J)	10.0168	10.0185
		Work (J)	1.4948	1.3321
		Max velocity disturbance (m/s)	-0.06	0
Mass-modified	Unconstrained	Energy bound (J)	✗	10.2146
		Work (J)	✗	1.3110
		Max velocity disturbance (m/s)	✗	-0.05
	PF-constrained	Energy bound (J)	✗	10.0618
		Work (J)	✗	1.4811
		Max velocity disturbance (m/s)	✗	-0.02

7. DISCUSSION

For the reference case, it can be seen in the trajectory phase plots, for both the unconstrained and PF-constrained parameterization shown in Figure 5, that the controlled trajectory perfectly follows the ideal trajectory. It can be seen in the corresponding torque and energy plots in Figure 6, that no actuator torques are generated and the energy tracks that of the unactated ideal case, as shown in Figure 4. It can be seen in Table 1 that both controllers have the same energy bound and maximum disturbance rejection as the unactuated ideal case. This serves as a validation for both the impedance controllers derived using inverse dynamics and least squares optimization.

For the slope-modified case, the initial impedance controllers, for both PF-constrained and unconstrained parameterization, allow the biped to achieve an LC gait on a flat surface ($\gamma = 0$) as can be seen in the trajectory phase plots in Figure 7. It can be seen in Table 1 that the velocity disturbance rejections are comparable to the ideal SWM, however, the energy bound is higher than the ideal case for both controllers. The work done by the actuators is similar for both controllers, however, it is almost 100 times the work done by gravity in the ideal case.

As can be seen in Table 1, RL of the initial impedance controllers for the slope-modified case increases the energy bound for both controllers, while decreasing the work done by the actuators. RL also leads to decreased disturbance rejection. As can be seen in Figures 9, the performance of the unconstrained parameterization levels off before the PF-constrained parameterization, indicating the unconstrained parameterization achieves a higher performance with less episodes compared to the PF-constrained parameterization.

For the mass-modified case, the initial impedance controllers, for both PF-constrained and unconstrained parameterizations, do not allow the biped to achieve an LC gait. This can be seen in the trajectory phase plots in Figure 12. The impedance controllers derived from inverse dynamics appear not to be able to compensate for the modified dynamics of the model.

However, RL of these initial policies allows the biped to achieve an LC gait as shown in Figure 14. This validates the use of RL for achieving an LC gait. As can be seen in Table 1, for both controllers the energy bound and work done is greater than the ideal case. While the robustness of the unconstrained controller is comparable to the ideal case, it is reduced for the PF-constrained controller. As can be seen in Figures 13, the performance of the unconstrained parameterization levels off before the PF-constrained parameterization.

For all cases, the energy bound and work done by the actuators was similar for both the PF-constrained and unconstrained controllers. As the implementation of the RL did not converge to a single optimal solution, the variance in the resulting energy and work was too large to draw an accurate comparison.

For all cases, there are no improvements to the robustness of the limit-cycle against velocity disturbances. The reason for this is that the episode (consisting of H steps of the limit-cycle) is a black-box from the perspective of the episode-based RL. Learning is based only on the inputs and outputs of the episode, therefore any unknown disturbances throughout the episode are not accounted for, and consequently the robustness is not improved by the RL. Exploring and learning throughout the episode may be one way to improve the robustness. Additionally, learning could take place in an unknown environment with unknown disturbances.

The scope of these results is limited by the variables of the simple walking model used. The only modifications tested were the ratio of the hip mass to foot mass, and the slope γ.

An interesting observation is the learned behavior of "swing-leg retraction" seen in the learned policy for both cases, as shown in Figures 10 and 14 . This is when the swing leg retracts at the end of a step until it hits the ground. It has been shown in Hobbelen and Wisse (2008) that swing-leg retraction can improve disturbance rejection.

8. CONCLUSION AND FUTURE WORK

In this work we successfully combined potential field control and reinforcement learning to achieve limit-cycle walking for a simple walking model. A limit-cycle was achieved on flat ground, and for a modified hip to foot mass ratio. The results demonstrate that a potential field controller can not only "emulate" the effect of gravity on the simple walking model, but also improve its performance if reinforcement learning is applied. The potential field-constrained controller provides safety by bounding the energy while performing equally well compared to an unconstrained controller. The performance of the RL leveled off faster for the unconstrained case.

Achieving a limit cycle gait on a SMW is trivial compared to more complex models. In future work the method presented in this paper could be applied to higher degree of freedom models. A strength of this method is the ability to bound the energy of the controlled system. In future work it could be explored how to enforce a desired energy bound. Improved tuning of the RL exploration and evaluation strategy could lead to improved policies and more conclusive results for the comparison of the unconstrained and PF-constrained parameterizations. More advanced RL methods could lead to potential fields that further improve performance and even increase robustness.

REFERENCES

Asano, F. and Yamakita, M. (2001). Virtual gravity and coupling control for robotic gait synthesis. *Systems, Man, and Cybernetics Part A: Systems and Humans, IEEE Transactions on*, 31(6), 737–745.

Deisenroth, M.P., Neumann, G., and Peters, J. (2011). A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2, 1–142.

Garcia, M., Chatterjee, A., Ruina, A., and Coleman, M. (1998). The Simplest Walking Model: Stability, Complexity, and Scaling. *Journal of Biomechanical Engineering*, 120(2), 281–288.

Hobbelen, D.G.E. and Wisse, M. (2007). Limit Cycle Walking. In M. Hackel (ed.), *Humanoid Robots: Human-like Machines*, June, pages 642–659. Vienna, Austria.

Hobbelen, D.G. and Wisse, M. (2008). Swing-leg retraction for limit cycle walkers improves disturbance rejection. *Robotics, IEEE Transactions on*, 24(2), 377–389.

Hogan, N. (1984). Impedance control: An approach to manipulation. In *American Control Conference, 1984*, 304–313. IEEE.

Hyon, S.H. and Cheng, G. (2006). Passivity-based full-body force control for humanoids and application to dynamic balancing and locomotion. *IEEE International Conference on Intelligent Robots and Systems*, 1, 4915–4922.

Kober, J., Bagnell, J.A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32, 1238–1274.

Kober, J. and Peters, J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, 84(1-2), 171–203.

Koditschek, D.E. (1987). Exact robot navigation by means of potential functions: Some topological considerations. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, 1–6. IEEE.

McGeer, T. (1990). Passive Dynamic Walking. *The International Journal of Robotics Research*, 9(2), 62–82.

Papageorgiou, M. (2012). *Optimierung: statische, dynamische, stochastische Verfahren.* Springer-Verlag.

Tedrake, R., Zhang, T., and Seung, H. (2004). Stochastic policy gradient reinforcement learning on a simple 3D biped. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3.

Vallery, H., Duschau-Wicke, A., and Riener, R. (2009a). Generalized elasticities improve patient-cooperative control of rehabilitation robots. In *Rehabilitation Robotics, 2009. ICORR 2009. IEEE International Conference on*, 535–541. IEEE.

Vallery, H., Duschau-Wicke, A., and Riener, R. (2009b). Optimized passive dynamics improve transparency of haptic devices. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 301–306. IEEE.

Appendix A

Lyapunov Stability Theory

Lyapunov stability means that solutions starting “close" to the equilibrium (within a distance δ) will remain “close" forever. Asymptotic stability means that solutions that start close to the equilibrium will eventually converge to the equilibrium.

For a nonlinear dynamical system

$$\dot{x} = f(x(t)), \quad x(0) = x_0, \tag{A-1}$$

where $x(t) \in \mathcal{D} \subseteq \mathbb{R}^n$ denotes the state vector, $\mathcal{D}$ an open set containing the origin, and $f : \mathcal{D} \to \mathbb{R}^n$ continuous on $\mathcal{D}$. If f has an equilibrium at x_e such that $f(x_e) = 0$ then

1. This equilibrium is said to be Lyapunov stable, if, for every $\epsilon > 0$, there exists a $\delta > 0$ such that, if $\|x(0) - x_e\| < \delta$, then for every $t \geq 0$ we have$\|x(t) - x_e\| < \epsilon$.

2. The equilibrium of the above system is said to be asymptotically stable if it is Lyapunov stable and there exists $\delta > 0$ such that if $\|x(0) - x_e\| < \delta$, then $\lim_{t\to\infty} \|x(t) - x_e\| = 0$.

Lyapunov's second method for stability, which is universally used, makes use of a Lyapunov function $V(\boldsymbol{x})$ which is analogous to the potential function of classical dynamics. For a system having a point of equilibrium at $x = 0$. Consider a function $V(x) : \mathbb{R}^n \to \mathbb{R}$ such that

$V(x) \geq 0$ with equality if and only if $x = 0$ (positive definite)
$\dot{V}(x) = \frac{d}{dt}V(x) \leq 0$ with equality not constrained to only $x = 0$ (negative semidefinite). For asymptotic stability, $\dot{V}(\boldsymbol{x})$ is required to be negative definite.

Then $V(\boldsymbol{x})$ is called a Lyapunov function candidate and the system is Lyapunov stable.

Appendix B

Simplest Walking Model

A diagram of the SWM is shown in Figure D-1.

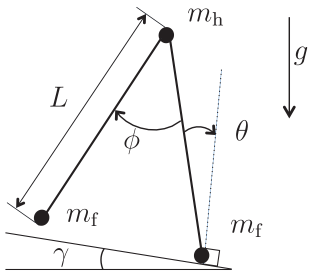

Figure B-1: Diagram of simplest walking model

The model consists of two massless rigid links of length L connected at the hip by a frictionless hinge. The mass is distributed over three point masses at the hip and feet such that the hip mass m_h is much larger than the foot mass m_f. The model is situated on a slope of angle γ and acts only under the force of gravity g. The configuration of the model is given by the ankle angle θ and hip angle ϕ. The equations of motion are derived using the same method presented in [19].

B-1 Equations of motion

The generalized coordinates are

$$\boldsymbol{q}(t) = \Big(x_{\mathrm{c}}(t), y_{\mathrm{c}}(t), \theta(t), \phi(t)\Big)^T \tag{B-1}$$

where the subscripts "c" denotes the contact point of the stance foot with the ground. A vector of the global coordinates of the point masses is

$$\boldsymbol{p} = (x_{\mathrm{st}}, y_{\mathrm{st}}, x_{\mathrm{hip}}, y_{\mathrm{hip}}, x_{\mathrm{sw}}, y_{\mathrm{sw}})^T \tag{B-2}$$

where subscripts "st" and "sw" denote the stance leg and swing leg respectively and subscript "hip" denotes the hip.

The global mass matrix $\mathbf{M}$ is defined

$$\mathbf{M} = \mathrm{Diag}(m_\mathrm{f}, m_\mathrm{f}, m_\mathrm{h}, m_\mathrm{h}, m_\mathrm{f}, m_\mathrm{f}). \tag{B-3}$$

The applied forces are only given by gravity and are therefore defined

$$\boldsymbol{f}_g = \mathbf{M}[\sin\gamma, -\cos\gamma, \sin\gamma, -\cos\gamma, \sin\gamma, -\cos\gamma]^T. \tag{B-4}$$

The generalized coordinates can be transformed to Cartesian positions using transfer function $\boldsymbol{F}$:

$$\boldsymbol{p} = \boldsymbol{F}(\boldsymbol{q}) = \begin{pmatrix} \mathrm{x_c} \\ \mathrm{y_c} \\ \mathrm{x_c} - L\ \sin(\theta) \\ \mathrm{y_c} + L\ \cos(\theta) \\ \mathrm{x_c} - L\ \sin(\theta) - L\ \sin(\phi - \theta) \\ \mathrm{y_c} + L\ \cos(\theta) - L\ \cos(\phi - \theta) \end{pmatrix}. \tag{B-5}$$

The equations of motion can be found using the virtual power equation

$$\delta\dot{\boldsymbol{p}}^T[\boldsymbol{f} - \mathbf{M}\ddot{\boldsymbol{p}}] = 0 \tag{B-6}$$

where $\delta\dot{\boldsymbol{p}}$ are kinematically admissible virtual velocities.

Differentiating the transfer function gives

$$\dot{\boldsymbol{x}} = \mathbf{F}_{,\boldsymbol{q}}\dot{\boldsymbol{q}},\ \ \delta\dot{\boldsymbol{x}} = \mathbf{F}_{,\boldsymbol{q}}\delta\dot{\boldsymbol{q}},\ \ \text{and}\ \ \ddot{\boldsymbol{x}} = \mathbf{F}_{,\boldsymbol{q}}\ddot{\boldsymbol{q}} + \mathbf{F}_{,\boldsymbol{qq}}\dot{\boldsymbol{q}}\dot{\boldsymbol{q}} \tag{B-7}$$

where the subscript comma operator followed by $\boldsymbol{q}$ denotes partial derivative by $\boldsymbol{q}$.

The reduced equations of motion are derived by the virutal power balance by substituting Equations B-7 into the virtual power equation B-6 and adding the contributions, $\delta\dot{\boldsymbol{q}}^T\boldsymbol{Q}$, from the generalized forces

$$\boldsymbol{Q} = (Q_{x_c}, Q_{y_c}, Q_\theta, Q_\phi)^T. \tag{B-8}$$

to the left hand side. This yields

$$[\mathbf{F}_{,\boldsymbol{q}}^T\mathbf{M}\mathbf{F}_{,\boldsymbol{q}}]\ddot{\boldsymbol{q}} = \mathbf{F}_{,\boldsymbol{q}}^T[\boldsymbol{f}_g - \mathbf{M}\mathbf{F}_{,\boldsymbol{qq}}\dot{\boldsymbol{q}}\dot{\boldsymbol{q}}] + \boldsymbol{Q}. \tag{B-9}$$

This equation can be written in terms of the reduced mass matrix and force vector

$$\bar{\mathbf{M}} = [\mathbf{F}_{,\boldsymbol{q}}^T\mathbf{M}\mathbf{F}_{,\boldsymbol{q}}],\ \ \bar{\boldsymbol{f}} = \mathbf{F}_{,\boldsymbol{q}}^T[\boldsymbol{f}_g - \mathbf{M}\mathbf{F}_{,\boldsymbol{qq}}\dot{\boldsymbol{q}}\dot{\boldsymbol{q}}] + \boldsymbol{Q} \tag{B-10}$$

resulting in

$$\bar{\mathbf{M}}\ddot{\boldsymbol{q}} = \bar{\boldsymbol{f}}. \tag{B-11}$$

For the unactuated case, Q_θ and Q_ϕ both equal zero. The contact forces at the stance foot Q_{x_c} and Q_{y_c} are only valid for $Q_{y_c} > 0$ given the contact condition $(x_c, y_c) = (0, 0)$ which defines the boundary condition. The equation of motion can be solved for the unknown accelerations and contact forces.

B-2 Impact equations

When the swing leg makes contact with the ground, referred to as heel strike, it is assumed to be a fully inelastic impact where no slip or bounce occurs. This corresponds to a coefficient of restitution of $e = 0$. In the case of a fully elastic impact, $e = 1$.

It is also assumed that there is an instantaneous moment of double stance, during which the swing leg becomes constrained and the the former stance leg becomes the swing leg. This is defined by the contact function

$$\boldsymbol{C}(\boldsymbol{q}) = \begin{bmatrix} x_{\text{sw}} \\ y_{\text{sw}} \end{bmatrix} = \begin{bmatrix} x_c - L\sin(\theta) - L\sin(\phi - \theta) \\ y_c + L\cos(\phi) - L\cos(\phi - \theta) \end{bmatrix}. \tag{B-12}$$

To account for the contact dynamics the generalized contact forces are introduced into the equations of motion using the Lagrangian multipliers $\boldsymbol{\lambda}$ corresponding to the contact velocities $\dot{\boldsymbol{C}}$, yielding

$$\bar{\mathbf{M}}\ddot{\boldsymbol{q}} + \boldsymbol{C}_{,\boldsymbol{q}}^T\boldsymbol{\lambda} = \bar{\boldsymbol{f}}. \tag{B-13}$$

Contact impulses, defined by

$$\boldsymbol{\rho} = \lim_{t^- \to t^+} \int_{t^-}^{t^+} \boldsymbol{\lambda} \mathrm{d}t \tag{B-14}$$

can be introduced into the problem by integrating equations of motion B-13 over the time of impact and taking the limit case, yielding

$$\lim_{t^- \to t^+} \int_{t^-}^{t^+} (\bar{\mathbf{M}}\ddot{\boldsymbol{q}} + \boldsymbol{C}_{,\boldsymbol{q}}^T\boldsymbol{\lambda})\mathrm{d}t = 0. \tag{B-15}$$

The reduced force vector $\bar{\boldsymbol{f}}$ disappears as it does not contain impulsive forces. Because the mass matrix remains constant during impact, the momentum equations become

$$\bar{\mathbf{M}}\dot{\boldsymbol{q}}^+ + \boldsymbol{C}_{,\boldsymbol{q}}^T\boldsymbol{\rho} = \bar{\mathbf{M}}\dot{\boldsymbol{q}}^- \tag{B-16}$$

where $\dot{\boldsymbol{q}}^+$ and $\dot{\boldsymbol{q}}^-$ are the velocities right before and after impact respectively. Newton's impact law, defined

$$\boldsymbol{C}_{,\boldsymbol{q}}\dot{\boldsymbol{q}}^+ = e\boldsymbol{C}_{,\boldsymbol{q}}\dot{\boldsymbol{q}}^- \tag{B-17}$$

can be used together with Equation B-16 to construct a set of linear equations

$$\begin{bmatrix} \bar{\mathbf{M}} & \boldsymbol{C}_{,\boldsymbol{q}}^T \\ \boldsymbol{C}_{,\boldsymbol{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}}^+ \\ \boldsymbol{\rho} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{M}}\dot{\boldsymbol{q}}^- \\ -e\boldsymbol{C}_{,\boldsymbol{q}}\dot{\boldsymbol{q}}^+ \end{bmatrix} \tag{B-18}$$

from which the velocities after impact and the contact impulses can be found.

A mapping is required to change the leg definitions and account for the reduction in velocity that occurs at impact from step h to step $h+1$. At double stance the leg angles are related by $\theta_h = 2\theta_a$. This results in the mapping

$$\boldsymbol{q}_{h+1} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & \cos(2\theta) & 0 & 0 \\ 0 & \cos(2\theta)(1 - \cos(2\theta) & 0 & 0 \end{bmatrix} \boldsymbol{q}_n. \tag{B-19}$$

B-3 Energy Equations

The total energy of a mechanical system is the sum of the potential energy, V, and kinetic energy, T:

$$E = V + T. \tag{B-20}$$

For the simple walking model, the potential energy, consisting only of the potential energy due to gravity, is equal to the object's mass, m, times the acceleration of gravity times the vertical height from a arbitrary zero position, h:

$$V = mgh. \tag{B-21}$$

$$V = m_f g L\Big(\cos(\theta-\gamma) + \cos(\theta-\gamma) - \cos(\phi-\theta+\gamma)\Big); \tag{B-22}$$

Kinetic energy, $T(\boldsymbol{q}, \dot{\boldsymbol{q}})$, is the energy of an object due to its motion, hence the dependency on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. For the SWM described in Cartesian space, the kinetic energy can be found

$$T = \frac{1}{2}\left(m_h\left(\dot{x}_h^2 + \dot{y}_h^2\right) + m_f\left(\dot{x}_f^2 + \dot{y}_f^2\right)\right) \tag{B-23}$$

where (x_h, y_h) is the coordinates of the hip and (x_f, y_f) are the coordinates of the swing foot and the velocities are defined

$$\dot{x}_h = -L\cos(\theta)\dot{\theta} \tag{B-24}$$
$$\dot{y}_h = -L\sin(\theta)\dot{\theta} \tag{B-25}$$
$$\dot{x}_f = -L\cos(\theta)\theta + L\cos(\phi-\theta)\dot{\theta} - L\cos(\phi-\theta)\dot{\phi} \tag{B-26}$$
$$\dot{y}_f = -L\sin(\theta)\dot{\theta} - L\sin(\phi-\theta)\dot{\theta} + L\sin(\phi-\theta)\dot{\phi}. \tag{B-27}$$

Appendix C

Least Squares Optimization

C-1 Weighted Recursive Least Squares Method

The procedure for recursive least squares given by [18] was modified to include weighing of various parameters such as training data, joints, and torque magnitude.

The least squares estimate of $\boldsymbol{w}$, denoted $\hat{\boldsymbol{w}}$ can be formulated as the minimization problem

$$\min_{\hat{\boldsymbol{w}}} \|\boldsymbol{G}(q_{\mathrm{d}})\hat{\boldsymbol{w}} - \boldsymbol{\tau}_{\mathrm{need}}\|^2_{\mathbf{Q}} \tag{C-1}$$

which is dependent on the number of training samples. The symmetric positive definite weighting matrix $\mathbf{Q}$ contains weights the reflect the importance of certain joints or training points

The recursive procedure for updating the estimate values of $\boldsymbol{w}$ is summarized as follows:

Initialization: The procedure is initialized with pseudo-training points and the parameter vector is initialized as a zero vector. This dummy initialization provides a regularization set for the LS problem.

Recursion: Collect the K measurements $\boldsymbol{\tau}_{\mathrm{train},i}$ with independent configuration vectors $\boldsymbol{q}_{\mathrm{train},i}$, for $i = 1, ..., K$.

Compute with each measurement $\boldsymbol{\tau}_{\mathrm{train},i+1}$ and associated model vector $\boldsymbol{q}_{\mathrm{train},i+1}$ the updated estimate value $\hat{\boldsymbol{w}}_{i+1}$ using the recursive update:

$$\boldsymbol{h}_k = \frac{\boldsymbol{\Pi}_k \boldsymbol{G}(\boldsymbol{q}_{k+1})}{\boldsymbol{G}(\boldsymbol{q}_{k+1})\boldsymbol{\Pi}_k \boldsymbol{G}(\boldsymbol{q}_{k+1})^T + (\boldsymbol{q}_{\mathrm{train},k+1})^{-1}} \tag{C-2}$$

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{h}_k(\boldsymbol{\tau}_{\mathrm{need},k+1} - \boldsymbol{G}(\boldsymbol{q}_{k+1})\hat{\boldsymbol{w}}_k) \tag{C-3}$$

$$\boldsymbol{\Pi}_{k+1} = \boldsymbol{\Pi}_k - \boldsymbol{h}_k \boldsymbol{G}(\boldsymbol{q}_{\mathrm{train},k+1})\boldsymbol{\Pi}_k. \tag{C-4}$$

The final estimate of $\boldsymbol{w}$ is used to define the potential field control law

$$\boldsymbol{\tau}_{\mathrm{fit}}(\boldsymbol{q}) = -\left(\frac{\mathrm{d}\boldsymbol{g}(\boldsymbol{q})}{\mathrm{d}\boldsymbol{q}}\right)^T \boldsymbol{w} \tag{C-5}$$

The recursive least-squares method is summarized in Algorithm 1.

Algorithm 1 Recursive Least-Squares

```
**Initialize:** Generate pseudo-training points
$\boldsymbol{\lambda}_0 = 0.1 \min(\boldsymbol{q}_{\text{train},\lambda})$
$\hat{\boldsymbol{w}}_0 = \boldsymbol{0}$
$\boldsymbol{\Pi}_0 = \frac{1}{\lambda_k \mathbf{I}}$
**for** $i = 1$ to $i = n$ **do**
    $\boldsymbol{h}_k = \frac{\boldsymbol{\Pi}_k \boldsymbol{G}(\boldsymbol{q}_{k+1})}{\boldsymbol{G}(\boldsymbol{q}_{k+1})\boldsymbol{\Pi}_k \boldsymbol{G}(\boldsymbol{q}_{k+1})^T + (\boldsymbol{q}_{\text{train},k+1})^{-1}}$
    $\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{h}_k(\boldsymbol{\tau}_{\text{need},k+1} - \boldsymbol{G}(\boldsymbol{q}_{k+1})\hat{\boldsymbol{w}}_k)$
    $\boldsymbol{\Pi}_{k+1} = \boldsymbol{\Pi}_k - \boldsymbol{h}_k \boldsymbol{G}(\boldsymbol{q}_{\text{train},k+1})\boldsymbol{\Pi}_k$
**end for**
$\boldsymbol{w} = \hat{\boldsymbol{w}}_n$
```

C-2 Radial Basis Functions

C-2-1 Compactly supported RBFs of minimal degree

The recursive least squares method is a multivariate interpolation using radial basis functions. Compactly supported radial basis functions are used to minimize the number of centers required. This results in a more efficient algorithm for computing and evaluating the interpolates. These functions are of the form:

$$f(\boldsymbol{r}) = \begin{cases} p(\boldsymbol{r}) & 0 \leq \boldsymbol{r} \leq 1, \\ 0 & \boldsymbol{r} > 1, \end{cases} \tag{C-6}$$

with a univariate polynomial

$$p(\boldsymbol{r}) = \sum_{j=1}^{N} c_j \boldsymbol{r}^j \tag{C-7}$$

where N is the degree of f, $c_N \neq 0$ and $\boldsymbol{r}$ is the radius function defined

$$r_k(\boldsymbol{q}) = s \, \|\boldsymbol{\delta}\| \tag{C-8}$$

where $k = 1$ to the number of samples and $\boldsymbol{\delta}$ is the distance vector defined

$$\boldsymbol{\delta}_k = \boldsymbol{q}_k - \boldsymbol{c}. \tag{C-9}$$

A class of functions of this form is given in [22]. For f to be continuous, positive definite on $\mathcal{R}^d$, then f must have degree $l \geq \lfloor d/2 \rfloor + 1$.

Since the simple walking model has two-degree of freedom ($d = 2$) the minimal degree of f must be $d = 3$. The function must have minimal smoothness of $C = 1$ because a smooth first derivative is required for the gradient of the radial basis functions to be used for interpolation. The function of this class that fits these requirements is

$$f(\boldsymbol{r}) = (1 - \boldsymbol{r})^4(4\boldsymbol{r} + 1). \tag{C-10}$$

C-2-2 Potential Function Derivation

The potential function can be estimated using normalized radial basis functions of the form

$$\psi(\boldsymbol{q}) = \frac{\sum_{i=1}^{K} w_i f_i(\boldsymbol{r})}{\sum_{j=1}^{K} f_j(\boldsymbol{r})} = \frac{f(\boldsymbol{r})\boldsymbol{w}}{\sum f(\boldsymbol{r})} \tag{C-11}$$

where K is the number or radial basis functions.

The control law given by the gradient of the potential function

$$\boldsymbol{\tau}_{\text{fit}}(\boldsymbol{q}) = -\nabla_q \psi(\boldsymbol{q}) \tag{C-12}$$

can be estimated as the transposed Jacobian

$$\boldsymbol{\tau}_{\text{fit}}(\boldsymbol{q}) = \frac{-1}{\left(\sum f[r(\boldsymbol{q})]\right)^2} \begin{bmatrix} \frac{\partial}{\partial q_1} f[r(\boldsymbol{q})] \sum f[r(\boldsymbol{q})] - f[r(\boldsymbol{q})] \sum \frac{\partial}{\partial q_1} f[r(\boldsymbol{q})] \\ \frac{\partial}{\partial q_2} f[r(\boldsymbol{q})] \sum f[r(\boldsymbol{q})] - f[r(\boldsymbol{q})] \sum \frac{\partial}{\partial q_2} f[r(\boldsymbol{q})] \end{bmatrix} \boldsymbol{w}. \tag{C-13}$$

Using C-10 as the radial basis function, it follows that

$$\frac{\partial}{\partial q_1} f[r(\boldsymbol{q})] = -20(1 - r(\boldsymbol{q}))^3 \cdot r(\boldsymbol{q}) \cdot \frac{\partial}{\partial q_1} r(\boldsymbol{q}) \tag{C-14}$$

$$\frac{\partial}{\partial q_2} f[r(\boldsymbol{q})] = -20(1 - r(\boldsymbol{q}))^3 \cdot r(\boldsymbol{q}) \cdot \frac{\partial}{\partial q_2} r(\boldsymbol{q}) \tag{C-15}$$

where

$$\frac{\partial}{\partial q_1} r(\boldsymbol{q}) = \frac{s\boldsymbol{\delta}_1}{\|\boldsymbol{\delta}\|} \tag{C-16}$$

$$\frac{\partial}{\partial q_2} r(\boldsymbol{q}) = \frac{s\boldsymbol{\delta}_2}{\|\boldsymbol{\delta}\|}. \tag{C-17}$$

Substituting equations C-8 and C-14 - C-17 into C-13 yields:

$$\boldsymbol{\tau}_{\text{fit}}(\boldsymbol{q}) = \frac{20s^2}{(\sum f[r(\boldsymbol{q})])^2} \begin{bmatrix} (1 - r(\boldsymbol{q}))^3 \boldsymbol{\delta}_1 \sum f[r(\boldsymbol{q})] - f[r(\boldsymbol{q})] \sum \left((1 - r(\boldsymbol{q}))^3 \boldsymbol{\delta}_1\right) \\ (1 - r(\boldsymbol{q}))^3 \boldsymbol{\delta}_2 \sum f[r(\boldsymbol{q})] - f[r(\boldsymbol{q})] \sum \left((1 - r(\boldsymbol{q}))^3 \boldsymbol{\delta}_2\right) \end{bmatrix} \boldsymbol{w}. \tag{C-18}$$

Appendix D

MatLab Code

This appendix includes selected MatLab code to show the implementation of this work's main contribution.

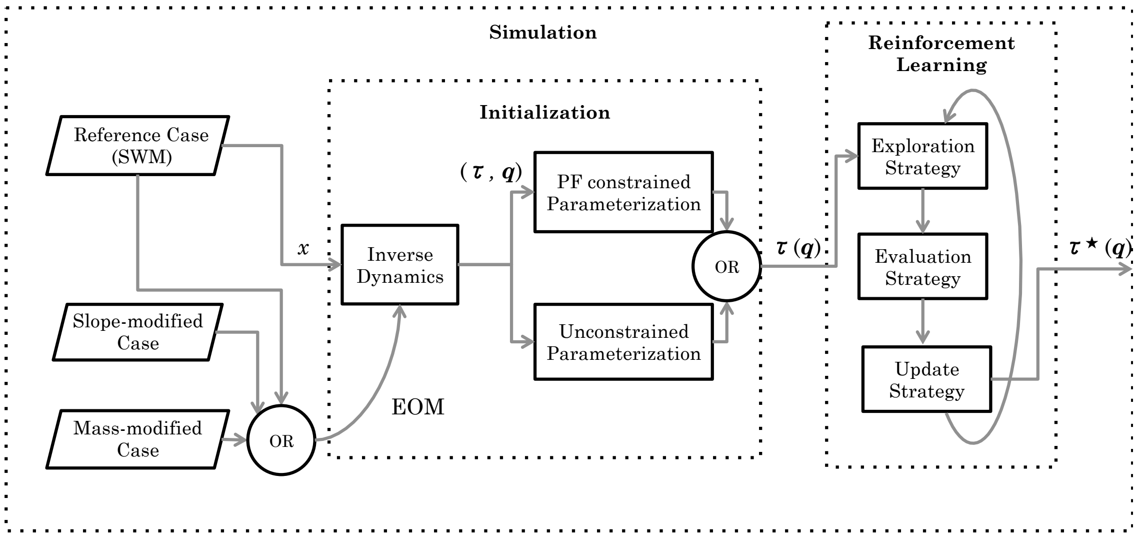

Figure D-1: Block diagram of methodology to show implementation

The outer block represents the main script given in Appendix D-1. The mcode relevant to the implementation of the Initialization block is given in Appendix D-2. The mcode relevant to the implementation of the Reinforcement Learning block is given in Appendix D-3.

D-1 Main script

The main script specifies the simulation, controller and data processing options and calls the corresponding functions. The main script is shown in the code block below.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  Simplest Walking Model Simulation:                    %
%      Least squares optimization + Reinforcment Learning Environment    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Created by Denise Feirstein based on SWM_simulation.m by Joost van der
% Weijde. Generates initial impedance control (unconstrained and
% PF-constrained) for Simplest Walking Model (SWM) using inverse dyanamics
% and recursive least squares (LS). Episode based policy-search
% Reinforcement Learning (RL) is used to improve the inital policy.

clc; clear all; clear global; %close all;

addpath(genpath('./EOM'));  addpath(genpath('./Integration'))
addpath(genpath('./Data')); addpath(genpath('./Data_processing'))
addpath(genpath('./RL'));   addpath(genpath('./Initialization'));
global par control_log RL_log xbasin

rng('default') % Initalize random number generator
%% Save Options
%-------------------------------------------
par.save_workspace = 1; % Save Data
par.save_location  = './Data/Results';
%-------------------------------------------
%% Simulation options
%-------------------------------------------
par.n_steps = 10; % number of consecutive steps
x0          = [0.1534  0.3068  -0.4942  -0.0231]; %initial conf.
par.x0      = x0;
par.ts      = 0.01; % time step
par.t_SWM   = 1.2180; % LC step time of ideal SWM for given x0
par.suppress_comments = 1;
% Velcoity disturbance options:
par.disturbance.which = 1; % choose 1 for stance leg, 2 for swing leg
par.disturbance.step  = 0;
par.disturbance.mag   = -0.05; % max no control: -0.05
%-------------------------------------------
% Set Model parameters:
par.modified_SWM        = 1; % Modify Model
par.modified_SWM_mod0   = 1; % on flat ground (gamma=0)
par.modified_SWM_mod10  = 0; % on flat ground, increase leg mass x10
par.modified_SWM_mod100 = 0; % on flat ground, increase leg mass x100
SWM_parameters;
%-------------------------------------------
%% Controller Options
%-------------------------------------------
par.controller_Unconstrained = 0;
par.controller_PFconstrained = 1;
%-------------------------------------------
% Load controller parameters:
par.load_w0                 = 'PF_Mod0_50_w0'; % must match controller
par.n_RBF                   = 50; % specify number of RBFs
%-------------------------------------------
% or derive controller parameters:
```

```
par.derive_controller           = 0;
par.n_RBF                       = 50; % choose number of RBFs
% Select ideal training data for LS:
par.Load_SWM_basin_training     = 1;
par.Load_SWM_training           = 0;
par.Generate_SWM_basin_training = 0; % Generate basin of attraction
% Set RBF parameters:
par.weightings   = 0;
par.support_base = 10;
RBF_parameters
%--------------------------------------------
% check to prevent conflict
if par.controller_PFconstrained && par.controller_Unconstrained
    fprintf('Controller conflict found'); return
end
%--------------------------------------------
% Reinforcement Learning
par.RL_PoWER            = 0; % choose update strategy
% Set RL Parameters:
if par.RL_PoWER
    par.prior_knowledge = 0;    % if 0 set par.w0 to zeros
    par.n_average       = 1;       % number of trials
    par.n_outer         = 5;    % number of learning episodes
    % Exploration Strategy
    par.n_batch_iter  = 100;     % exploration batch size
    par.variance_init = 1.e-7;  % intial exploration variance
    par.variance_end  = 1.e-8; % end exploratio variance
    % Evaluation Strategy:
    % R = max(0,R_step - R_change*(dangle^2+dvelocity^2)...
    %                   - R_fallen - R_tau*RL_log.R_tau);
    par.R_step     = 1;   % reward step
    par.R_change   = 10;  % penalize change in step angle and velocity
    par.R_tau_ank  = 10;  % penalize ankle torques
    par.R_tau_hip  = 100; % penalize hip torques
    par.R_t        = 1;   % penalize change in LC step time for ideal
    par.mimic_SWM  = 1;   % if 0 R_step = -R_step*10*sin(state(1))
end
%--------------------------------------------
% Virtual Gravity controller
par.controller_vg     = 0; % choose to implement virtual gravity control
if par.controller_vg
    par.v_gamma       = 0.000; % virutal slope
    par.gamma         = 0;
end
%--------------------------------------------
run_name % Set Run Name for saving
%--------------------------------------------
%% Plot options
%--------------------------------------------
par.p_animation = 0; par.range = [1 2]; % show animation for range of steps
par.p_RBFs      = 0; % plot RBF center locations and support base
par.p_RL        = 1; % plot RL performance
par.p_energy    = 1; % plot energy and control
par.p_poincare  = 0; % plot basin of attraction
par.p_vector_field = 1; % plot control as vector field
%--------------------------------------------
%% Calculate equations symbolically
%--------------------------------------------
```

```
% SWM_derivation_eom % (uncomment if necessary)
% SWM_derivation_ff
%--------------------------------------------------
%% Derive or load initial control policy
%--------------------------------------------------
T_need = [];
if par.derive_controller     % derive controller parameters
    w = initialization(par); % Least squares optimization
    cd(par.save_location);   % Save initial policy:
    save([par.run_name '_w0'],'w');cd('..');cd('..')
    par.w0 = w; par.w = w;
else                         % load controller parameters
    cd(par.save_location)
    load(par.load_w0);cd('..');cd('..')
    par.w0 = w; par.w = w;
end
%--------------------------------------------------
%% Integration options
%--------------------------------------------------
par.unidirectional_contact = 1; par.detect_faceplant = 1; par.refine = 4;
par.options = odeset('AbsTol',1e-6,'RelTol',1e-3,'Events',...
              @event_leftcontact,'InitialStep',par.ts,'Refine',par.refine);
par.dt_I = 1.6; %[s] maximum integration time to event
%--------------------------------------------------
%% Simulate steps
%--------------------------------------------------
if par.RL_PoWER %&& par.controller_PFconstrained
    RL_integration_sequence
elseif ~par.controller_PFconstrained && ~par.controller_Unconstrained
     w = []; % If no control
    [R_log,xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w);
else % If no RL:
    [R_log,xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w);
end
%--------------------------------------------------
%% Process data
%--------------------------------------------------
if par.RL_PoWER
    sD = sortrows(RL_log.D); % Verify best policy
    w  = sD(end,2:end)';
    [R_log,xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w);
end

process_data(tlog,xlog,T_need,x0,telog,par);
%--------------------------------------------------
%% Save data
%--------------------------------------------------
if par.save_workspace
    cd(par.save_location)
    save(par.run_name);cd('..');cd('..');%cd('..')
end
```

D-2 Initialization functions

The initialization is implemented in `initialization.m`. This function performs inverse dynamics and calls the least squares optimization through function `compact_NRBFs_recursive_Unconstrained_3order.m` or `compact_NRBFs_recursive_PFconstrained_3order.m` depending on the desired parameterization. The latter two functions are based on code provided by Heike Vallery, which was adapted to the 2 DoF case and translated from German to English.

D-2-1 `initialization.m`

```
function [w] = initialization(par)
% Created by Denise Feirstein
% This funciton generates/loads traing data then calls the recursive least
% squares optimization for the PF-constrained controller and outputs the
% corresponding parameter vector.

% Model Parameters
L     = par.L;
g     = par.g;
gamma = par.gamma;
mf    = par.mf;
mh    = par.mh;
xa    = 0;
ya    = 0;

%% Generate/Load Training Data
%-----------------------------------------------
% Generate Training data
%-----------------------------------------------
if par.Load_SWM_training
    load xlog_SWM       % load ideal LC trjectory
    traj = xlog_SWM;
elseif par.Load_SWM_basin_training
    load xbasin_final   % load trajectory of basin of attraction
    traj = xbasin;
end
samples = length(traj); % determine number of samples in training data

%% Inverse dynamics
% Generate desired torques
%-----------------------------------------------
T_need = zeros(samples,2);
if par.modified_SWM
    for i_sample = 1: samples
        % Ideal trajectory
        theta    = traj(i_sample,1); % ankle angle
        phi      = traj(i_sample,2); % hip angle
        theta_d  = traj(i_sample,3); % ankle angular velocity
        phi_d    = traj(i_sample,4); % hip angular velocity
        state    = traj(i_sample,:)';
        qdd      = SWM_qdd(state,0,0,par.gamma_swm); % EOM
        theta_dd = qdd(1);           % ankle angular acceleration
        phi_dd   = qdd(2);           % hip angular acceleration
        traj(i_sample,5) = qdd(1);
        traj(i_sample,6) = qdd(2);
        % Compute Torques for modified model using inverse dynamics
        [Torques] = SWM_Q_ff(L,g,gamma,mf,mh,xa,ya,...
            [theta phi theta_d phi_d theta_dd phi_dd]);
        T_need(i_sample,:) = Torques';
    end
end

%% Implements Recursive LS
%-------------------------------------------------------------
% Desired Trajectory:
q1 = traj(:,1); % ankle angle
q2 = traj(:,2); % hip angle
% Desired Torques:
T_d1 = T_need(:,1); % ankle torque
T_d2 = T_need(:,2); % hip torque

% Calculate weights using weighted recursive LS
if par.controller_PFconstrained
    [w] = compact_NRBFs_recursive_PFconstrained_3order(q1,q2,T_d1,T_d2,par);
elseif par.controller_Unconstrained
    [w1,w2] = compact_NRBFs_recursive_unconstrained_3order(q1,q2,T_d1,T_d2,
        par);
    w = [w1;w2];
end
end
```

D-2-2 compact_NRBFs_recursive_Unconstrained_3order.m

```
function [w1,w2] = compact_NRBFs_recursive_Unconstrained_3order(q1,q2,T_ges1
    ,T_ges2,par)
% Created by Denise Feirstein based on
% compact_NRBFs_recursive.m by Heike Vallery.
% Solution for tau = G * w is calculated recursively, were w = [w1;w2].
% G is the normalized RBF.
% The packet length determines how many data points are pushed through.
% The higher the value, the faster, but memory problems can occur.
% A compactly supported RBF with minimal degree d=3 and smoothness C=1 is
% chosen from Wendeland (1995).

%% RBF parameters:
c1_res = par.c1_res; % center point coordinates in dimension 1
c2_res = par.c2_res; % center point coordinates in dimension 2
radiusscale   = par.radiusscale; % determines size of RBF support
packet_length = par.packet_length;
% For weighted recursive LS
weightings    = par.weightings;
if weightings
    samples_p = par.samples_p; % indicate sample p in training data
    samples_e = par.samples_e; % indicate sample e in training data
    lambda_p  = par.lambda_p;  % sample p weight
    lambda_e  = par.lambda_e;  % sample e weight
    lambda_a  = par.lambda_a;  % ankle weight
    lambda_h  = par.lambda_h;  % hip weight
    lambda_w  = par.lambda_w;  % control torque magnitude weight
end
n_q = par.n_q;                 % DoF (number of generalize coordinates)
```

```
samples = length(q1);           % number of training samples
support_num = length(c1_res);  % number of RBFs
C = [c1_res,c2_res];           % RBF center locations

%% build q-vector, with or without weights:
%------------------------------------------
if weightings==1   % With various weights:
    % 1) weighting samples p versus samples e:
    weigh_p_e_vector=[ones(samples_p,1)*lambda_p;
                      ones(samples_e,1)*lambda_e];
    % 2) Condition for definiteness missing since only gradient information
    %    available. Arbitrarilly select Sum (w_i) = support number.
    sumw =   support_num;
    % 3) Give higher weight to lower torques: w = 1 for tau = 0,
    %    w = 1-lambda_w * tau / taumax
    q_vector = [lambda_a*weigh_p_e_vector.*(1-lambda_w*abs(T_ges1)...
                                              ./max(abs(T_ges1)));
                lambda_h*weigh_p_e_vector.*(1-lambda_w*abs(T_ges2)...
                                              ./max(abs(T_ges2)));
                1];
else % No weights
    sumw     = support_num; % only need equation (2) for definitiveness
    q_vector = ones(samples*2+1,1); % Initialize with equal weights
end

%% Recursive LS:
% Based on Gemaess Papageorgiou, "Optimierung", S.63:
%----------------------------------------------------
% Initialization:
%----------------------------------------------------
% Generate pseudo-training points on support points that have the value 0:
lambda_init = .1*min(q_vector);
C_n    = eye(support_num);
Y_n    = zeros(support_num,1);
Q_n    = lambda_init*eye(support_num);
xhat_n = zeros(support_num,1);          % equivalent to pinvQ(C_n,Q_n)*Y_n;
Pi_n   = 1/lambda_init*eye(support_num);% equivalent to: inv(C_n'*Q_n*C_n);
% set: k = n. n in this case is 0, since no training data initialized.
xdach_k_1 = xhat_n;
Pi_k_1    = Pi_n;
xdach_k_2 = xhat_n;
Pi_k_2    = Pi_n;
% To undo the influence of the initialization, make -lambda_init=q at end.

%% Recursion including data point calculation.:
%----------------------------------------------------
disp('start Data Loop')

for index_data = 1:packet_length:samples
    fprintf('Processing samples %d - %d of %d\n',...
             index_data, index_data+packet_length, samples);
    % determine indicies of datapoints in new packet:
    kdata             = index_data:min(index_data+packet_length-1,samples);
    act_packet_length = length(kdata);
    %------------------------------------------------------------
    % Calculate the distances of all the data points to all RBF centers:
    %------------------------------------------------------------
    delta  = []; % intialize delta vector
    radius = zeros(act_packet_length,support_num); % initialize radius
    for sample = kdata
        for centerpoint = 1:support_num
            delta = [q1(sample),q2(sample)]-C(centerpoint,:);
            radius(sample-index_data+1,centerpoint) = ...
                                                   radiusscale*norm(delta);
        end
    end
    R_sat  = min(radius,1); % Saturate radius to size of radiusscale
    %-----------------------------------------------------
    % Unconstrained parameterization:
    %-----------------------------------------------------
    fR     = (1-R_sat).^4.*(4*R_sat+1); % Compactly supported RBF
    sum_fR = (repmat(sum(fR,2),1,support_num)); % sum across rows

    % It then applies: tau = G * w
    G = fR./sum_fR; % normalized radial basis function
    %-----------------------------------------------------
    % prevent memory problems
    clear R_sat fR sum_fR

    % For Tau1:
    tau_1      = T_ges1(kdata);
    c_kplus1_1 = G;
    y_kplus1_1 = tau_1;
    q_kplus1_1 = diag(q_vector([kdata]));
    % For Tau2:
    tau_2      = T_ges2(kdata);
    c_kplus1_2 = G;
    y_kplus1_2 = tau_2;
    q_kplus1_2 = diag(q_vector([kdata+samples]));

    %---------------------------------------------------
    %Update LEAST SQUARES:
    %---------------------------------------------------
    % For Tau1:
    [xdach_kplus1_1,Pi_kplus1_1] = recursiveLS_update(xdach_k_1,Pi_k_1,...
                                         y_kplus1_1,q_kplus1_1,c_kplus1_1);
    xdach_k_1 = xdach_kplus1_1; % set next estimate
    Pi_k_1    = Pi_kplus1_1;
    % For Tau2:
    [xdach_kplus1_2,Pi_kplus1_2] = recursiveLS_update(xdach_k_2,Pi_k_2,...
                                        y_kplus1_2,q_kplus1_2,c_kplus1_2);
    xdach_k_2 = xdach_kplus1_2; % set next estimate
    Pi_k_2    = Pi_kplus1_2;
end
    disp('stop loop')

%% Undo the effect of initialization  by adding q = -lambda_init:
% If not sufficient training points, to produce a stable mesh,
% uncomment lines:
% c_kplus1 = C_n;
% y_kplus1 = Y_n;
% q_kplus1 = -lambda_init*diag(ones(anzahlstuetzen,1));
% % Last Update:
% [xdach_kplus1,Pi_kplus1]=recursiveLS_update(xdach_k,Pi_k,y_kplus1,...
%                                                       q_kplus1,c_kplus1);

```

```
%% Solution:
w1 = xdach_kplus1_1;
w2 = xdach_kplus1_2;
end
```

D-2-3 `compact_NRBFs_recursive_PFconstrained_3order.m`

The function `compact_NRBFs_recursive_PFconstrained_3order.m` is similar to function `compact_NRBFs_recursive_Unconstrained_3order.m` above with the exceptions of lines 95 though 116. Instead, they are given as:

```
    %-----------------------------------------------------------------
    % PF-constrained Parameterization:
    %-----------------------------------------------------------------
    fR = (1-R_sat).^4.*(4*R_sat+1); % minimal degree d=3, smoothness C=3
    % Potential Funciton: PF = sum(w*fR)/sum(fR)
    % Gradient PF: nabla_PF = G*w_vector, where
    %   G = 20*radiusscale^2 ./ sum_fR.^2 .*...
    %  [Delta_k_R13.*sum_fR - fR.*repmat(sum(Delta_k_R13,2),1,support_num)]
    Delta_1 = repmat(q1(kdata),1,support_num)...
              -repmat(c1_res',act_packet_length,1);
    Delta_2 = repmat(q2(kdata),1,support_num)...
              -repmat(c2_res',act_packet_length,1);
    sum_fR  = (repmat(sum(fR,2),1,support_num)); % sum across rows
    R1_3    = (1-R_sat).^3;
    Delta_1_R13 = Delta_1.*R1_3;
    Delta_2_R13 = Delta_2.*R1_3;
    radiusscale20sumfr2 = 20*radiusscale^2 ./ sum_fR.^2;
    % Therefore:
    G=[radiusscale20sumfr2.*(Delta_1_R13.*sum_fR - ...
                        fR.*repmat(sum(Delta_1_R13,2),1,support_num));
       radiusscale20sumfr2.*(Delta_2_R13.*sum_fR - ...
                        fR.*repmat(sum(Delta_2_R13,2),1,support_num))];
    %-----------------------------------------------------------------
```

D-3 Reinforcement Learning script and functions

The RL is performed though script RL_integration_sequence.m. This script iteratively calls the exploration strategy (Batch_exploration.m), update strategy (PoWER_update.m) and walking simulation integration (RL_integration_sequence.m). The evaluation strategy given in Policy_Eval.m is called from the integration function. The MATLAB function of the PoWER update algorithm is based on code from Jens Kober.

D-3-1 RL_integration_sequence.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    RL Integration Sequence    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Created by Denise Feirstein

global par RL_log

% Initialize RL storage variables:
RL_log.D                = [];
RL_log.mean_performance = [];

% Define RL Performance plot:
hf_performance = figure();
ha_performance = axis();
xlabel('episode (x 10)')
ylabel('average return')
title('Mean Performance')

for i_average = 1:par.n_average; % RL trials for calculating mean
    performance
    % set intial parameters:
    if par.prior_knowledge
        w = par.w0;
    else
        w = zeros(length(par.w0),1);
    end
%     if par.controller_PFconstrained
%         w = par.w0;
%     elseif par.controller_Unconstrained
%         w = [par.w1_0;par.w2_0];
%     end

    % Run initial policy
    [R_log,xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w);

    % Evaluate initial policy
    Return_0 = sum(R_log);
    RL_log.performance     = Return_0;  % Log initial performance
    RL_log.episode{1}.xlog = xlog;      % Log initial trajectory

    for i_outer = 1:par.n_outer % RL episodes
        current_param = w;

        % Exploration strategy:
        [s_Return,Return,param] = Batch_exploration(current_param,x0,i_outer
            ,par);
        s_Return = sortrows(s_Return);

        % Updated strategy:
        [current_param_plus1] = PoWER_Update(s_Return,Return,param,
            current_param,par);
        w = current_param_plus1;

        % Run updated policy without exploration
        [R_log,xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w);
        performance = sum(R_log); % Evaluate performance

        % Log performance
        RL_log.performance = [RL_log.performance;performance];
        RL_log.D = [RL_log.D; performance w']; % log parameters

        % Plot performance
        plot(RL_log.performance)
        xlabel('episode')
        ylabel('average return')
        title('Mean Performance: PF-constrained Policy')
        drawnow

%         disp(['Outer loop ' num2str(i_outer)])
    end
    % Log performance and final trajecgory of last trial:
    RL_log.mean_performance = [RL_log.mean_performance RL_log.performance];
    RL_log.episode{i_outer+1}.xlog = xlog;
end

%% Save final policy parameters:
if par.save_workspace
    cd(par.save_location)
    save([par.run_name '_w'],'w')
    cd('..');cd('..')
end
```

D-3-2 Batch_exploration.m

```
function [s_Return,Return,param] = Batch_exploration(current_param,x0,
    i_outer,par)
% Created by Denise Feirstein
% Perturn policy parameter vector independintly n_iter times

n_iter  = par.n_batch_iter;
n_RBF   = numel(current_param); % number of basis functions
n_outer = par.n_outer;

% set the exploration variance
variance_init = par.variance_init; % initial varience
variance_end  = par.variance_end;  % end varience
variance = variance_init + (variance_end-variance_init)*i_outer/n_outer;
variance = variance .*ones(n_RBF,1);

% Initialize RL storage variables
Return    = zeros(1,n_iter+1);
param     = zeros(n_RBF,n_iter+1);
```

```
s_Return = -1*ones(n_iter+1, 2);
param(:,1) = current_param;

parfor iter = 1:n_iter;
    % Explore around current parameter vector:
    w = current_param + variance.^(.5).*randn(n_RBF,1);

    param(:,iter) = w; % log exploration parameter vectors
    % Run exploraiton policy:
    [R_log,xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w);
    Return(iter)      = sum(R_log);           % Calculate return of episode
    s_Return(iter,:) = [Return(iter) iter]; % store return and iteration
end
% s_Return = sortrows(s_Return); % sort exploration iterations by return
end
```

D-3-3 `PoWER_update.m`

```
function [ current_param_plus1 ] = PoWER_Update(s_Return,Return,param,
    current_param ,par)
% Created by Denise Feirstein based on PoWER Algorithm (Kober 2011)
% Inputs:
% s_Return = top 10 peturbed interations with highest return
% param    = matrix of peterubed parameter vectors for all iterations
% current_param = last updated parameter vector

% update the policy parameters
param_nom  = zeros(numel(current_param),1);
param_dnom = 0;

% Take the 10 best episodes as importance sampling
for i = 1:10
    % get the episode number for the 10 best episodes
    j = s_Return(end+1-i,2);
    % calculate the exploration with respect to the current parameters
    temp_explore = (param(:,j)-current_param);
    % weight the exploration by the return:
    param_nom  = param_nom + temp_explore*Return(j);
    param_dnom = param_dnom + Return(j); % sum returns
end
% Update (the normalization is taken care of by the division):
param = current_param + param_nom./(param_dnom+1.e-10);
current_param_plus1 = param; % set the new mean of the parameters
end
```

D-3-4 `RL_integration_sequence.m`

```
function [R_log, xlog,tlog,xelog,telog] = integration_sequence_fun(par,x0,w)
% Created by Denise Feirstein
% based on integration_sequence.m by Joost van der Weijde

clear fallen % clear logs
if par.controller_PFconstrained
    par.w = w;
```

```
else
    par.w1 = w(1:length(w)/2);
    par.w2 = w(length(w)/2+1:end);
end

% Initialize logs
tlog          = [];    xlog  = [];
telog         = [];    xelog = [];
par.t_step_log = 0;    par.x_step_log = zeros(1,size(par.x0,2));
R_log         = [];
x0_internal   = par.x0;
xalog         = x0;
t0            = 0; %[s] starting time

for ii = 1:par.n_steps % simulate steps
    par.n_s = ii;
    par.x0_curr    = x0_internal(ii,:);

    % disturbance at start of step
    nr_disturbances = length(par.disturbance.step);
    for ii_dist = 1:nr_disturbances
        if ~isempty(par.disturbance.step(ii == par.disturbance.step))...
                && par.disturbance.which == 1
            x0_internal(ii,3)  = x0_internal(ii,3)*(1-par.disturbance.mag);
        elseif ~isempty(par.disturbance.step(ii==par.disturbance.step))...
                && par.disturbance.which == 4
            x0_internal(ii,4)  = x0_internal(ii,4)*(1-par.disturbance.mag);
        end
    end

    % integrate single step:
    [te,xe,t,x,xa,ie,fallen,par] = body_rightstep_fun(t0,...
                                                     x0_internal(ii,:),par);
    % log state, time and events:
    telog = [telog;te];
    xelog = [xelog;xe];
    tlog = [tlog;t];
    xlog = [xlog;x];
    t0   = te(end);
    par.t_step_log = [par.t_step_log;te(end)];
    par.x_step_log = [par.x_step_log;xe(end,:)];
    % set next initial state:
    x0_internal(ii+1,:) = set_init_state(xa(:,end).',par);

    if par.RL_PoWER  % Evalute step
        xalog = [xalog;xa'];
        par.fallen = fallen;
        [R] = Policy_Eval(x,xalog,x0,telog,ii,par);
        R_log = [R_log ; R];
    end

    if fallen % check for fall
        if ~par.suppress_comments
            fprintf('  Model has fallen.\n');
        end
        par.fallen = 1;
        par.range(2) = min(par.range(2),ii+1); % adjust animation
        break
    else
        par.fallen = 0;
    end
end
end
```

D-3-5 Policy_Eval.m

```
function [R] = Policy_Eval(state,xa_log,x0,telog,ii,par)
% Created by Denise Feirstein
% Evaluate policy using reward function

fallen    = par.fallen;
R_step    = par.R_step;
R_tau_hip = par.R_tau_hip;
R_tau_ank = par.R_tau_ank;
R_change  = par.R_change;
R_t       = par.R_t;

tstep  = [telog(1); telog(2:end)-telog(1:end-1)];
dtstep = norm(par.t_SWM*ones(length(tstep),1)-tstep);

R_fallen = 0;
% if fallen % (Uncomment to penalize fall)
%     R_fallen = par.R_fallen;
% end

if par.controller_PFconstrained
    [T_hip,T_ank] = PFconstrained_controller(state, par);
elseif par.controller_Unconstrained
    [T_hip,T_ank] = Unconstrained_controller(state, par);
end
Tau = sqrt( T_hip^2 + T_ank^2 );
Tau_hip = sqrt( T_hip^2 );
Tau_ank = sqrt( T_ank^2 );

if par.mimic_SWM
    % Calculate change in angle and angular velocity from SWM LC
    dangle    = xa_log(end,1) - par.x0(1);
    dvelocity = xa_log(end,3) - par.x0(3);
    par.R_change_implement = 0;
else
    % Calculate change in angle and angular velocity from previous step
    dangle    = xa_log(end,1) - xa_log(end-1,1);
    dvelocity = xa_log(end,3) - xa_log(end-1,3);
end

if par.RL_PoWER % PoWER update cannot handle negative rewards
    R = max(0,R_step - R_change*(dangle^2+dvelocity^2) - R_t*dtstep...
              - R_fallen - R_tau_hip*Tau_hip - R_tau_ank*Tau_ank);
else
    R = R_step - R_change*(dangle^2+dvelocity^2)...
        - R_fallen - R_tau*Tau;
end
end
```

Bibliography

[1] F. Asano and M. Yamakita. Virtual gravity and coupling control for robotic gait synthesis. *Systems, Man, and Cybernetics Part A: Systems and Humans, IEEE Transactions on*, 31(6):737–745, 2001.

[2] F. Asano, M. Yamakita, N. Kamamichi, and Z. W. Luo. A novel gait generation for biped walking robots based on mechanical energy constraint. *IEEE Transactions on Robotics and Automation*, 20(3):565–573, 2004.

[3] K. R. Chevva. *Practical challenges in the method of controlled Lagrangians.* PhD thesis, Virginia Polytechnic Institute and State University, 2005.

[4] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science (New York, N.Y.)*, 307(5712):1082–1085, 2005.

[5] T. De Boer. *Foot placement in robotic bipedal locomotion.* TU Delft, Delft University of Technology, 2012.

[6] V. Duindam, S. Stramigioli, and J. Scherpen. Passive compensation of nonlinear robot dynamics. *Robotics and Automation, IEEE Transactions on*, 20(3):480–488, 2004.

[7] M. Garcia, A. Chatterjee, A. Ruina, and M. Coleman. The Simplest Walking Model: Stability, Complexity, and Scaling. *Journal of Biomechanical Engineering*, 120(2):281–288, 1998.

[8] A. Goswami, B. Thuilot, and B. Espiau. A Study of the Passive Gait of a Compass-Like Biped Robot: Symmetry and Chaos. *The International Journal of Robotics Research*, 17(12):1282–1301, 1998.

[9] D. Hobbelen, T. De Boer, and M. Wisse. System overview of bipedal robots Flame and TUlip: Tailor-made for Limit Cycle Walking. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2486–2491, 2008.

[10] D. G. E. Hobbelen and M. Wisse. Limit Cycle Walking. In M. Hackel, editor, *Humanoid Robots: Human-like Machines*, number June, pages pages 642–659. Vienna, Austria, 2007.

[11] N. Hogan. Impedance control: An approach to manipulation. In *American Control Conference, 1984*, pages 304–313. IEEE, 1984.

[12] R. Katoh and M. Mori. Control method of biped locomotion giving asymptotic stability of trajectory. *Automatica*, 20(4):405–414, 1984.

[13] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 2013.

[14] M. Lapeyre, P. Rouanet, J. Grizou, S. Nguyen, F. Depraetre, A. Le Falher, and P.-Y. Oudeyer. Poppy project: Open-source fabrication of 3d printed humanoid robot for science, education and art. In *Digital Intelligence 2014*, page 6, 2014.

[15] T. McGeer. Passive Dynamic Walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[16] Y. Ogura, H. Aikawa, H.-o. Lim, and A. Takanishi. Development of a human-like walking robot having two 7-dof legs and a 2-dof waist. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 134–139. IEEE, 2004.

[17] R. Ortega, A. J. van der Schaft, I. Mareels, and B. Maschke. Putting energy back in control. *IEEE Control Systems Magazine*, 21(2):18–33, 2001.

[18] M. Papageorgiou. *Optimierung: statische, dynamische, stochastische Verfahren*. Springer-Verlag, 2012.

[19] A. Schwab and M. Wisse. Basin of attraction of the simplest walking model. In *Proceedings of the ASME design engineering technical conference*, volume 6, pages 531–539, 2001.

[20] M. W. Spong. Passivity based control of the compass gait biped. *Proc. of IFAC World Congress, Beijing, China*, pages 19–24, 1999.

[21] R. Tedrake, T. Zhang, and H. Seung. Stochastic policy gradient reinforcement learning on a simple 3D biped. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3, 2004.

[22] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, 1995.