Framework for Analysing Predictive Controllers for Synthesis of Biologically Inspired Simulation of Human Gait

Jakob Welner
s124305

January 2016

B.Sc Thesis, 20 ECTS
Supervisor: Christensen, D. J.*
Co-Supervisor: Alkjær, T.**
Co-Supervisor: Sørensen, H.***

Department of Mechanical Engineering
The Technical University of Denmark

DTU

Made in Denmark by
Technical University of Denmark

*Associate Professor, Electrical Engineering, Technical University of Denmark
**Associate Professor, Motor Control Lab, Copenhagen University
***Associate Professor, Department of Public Health - Sport Science, Aarhus University

Department of Mechanical Engineering
Technical University of Denmark
Produktionstorvet, Building 427
DK-2800 Kgs. Lyngby
Denmark

http://www.mek.dtu.dk/
Tel: (+45) 45 25 19 60
Fax: (+45) 45 93 14 75
E-mail: info@mek.dtu.dk

Publication Reference Data
Welner, J.
Framework for Analysing Predictive Controllers for Synthesis of Biologically Inspired Simulation of Human Gait
B.Sc Thesis
Technical University of Denmark, Mechanical Engineering
January 2016

Abstract

Medical research has long strived to gain an insight into the inner workings of human body. Making light on what is going on behind the scenes of the body's functions enables to better diagnose, treat and to get inspired by the complicated mechanism that nature has created and optimized through millions of years. The study presented in this thesis dives into the multidisciplinary topic of Predictive Simulation, a method that can be applied to synthesize human-like gait patterns. This is done by establishing a bottom-up model of the human locomotor and optimizing it for high-level objectives such as minimum effort per distance travelled. The topic of Predictive Simulation is explained through a close-up of its main subsystems: Mechanical Model, Predictive Controller, Simulator, Optimizer and Objective Function. The freely available software package *PredictiveSim* by Dorn et al. (2015) forms the basis of this study, both through a series of experiments but also as a platform to extend upon. *PredictiveSim* is a ready-made functional system that combine all components necessary for reaching a state-of-the-art human-like gait pattern. The software relies on a reflex-based parametric controller with 77 design variables, which relate sensory input to muscle activation. When the parameters are optimized these enable the system to perform continuous autonomous walking with inherent balance. The software is studied at first and subsequently extended in order to increase its functionality. The new functions include Optimization with Perturbation, Supraspinal Control by varying parameters during simulations and the inclusion of Blind Random Search as an alternative optimization algorithm. The experiments are performed through a series of optimizations and simulations with a range of varying speeds and perturbations. The resulting design parameters are analysed using Machine Learning and Principal Component Analysis to investigate potential underlying patterns. Several tendencies are found which suggests that systems like these, while normally requiring large amounts of computing power for optimizing variables, may benefit from applying Data Mining techniques to approximate or predict solutions beforehand.

– This page intentionally left almost blank –

Synopsis (in Danish)

Medicinsk forskning har længe søgt insigt i de indre mekanismer i kroppen. Denne indsigt gør det muligt at udføre bedre diagnoser, behandle og blive inspireret a det komplekse maskineri som naturen har skabt og optimieret gennem millioner af år. Projektet der præsenteres i denne opgave dykker ned i det tværfaglige emne om Predictive Simulation, en metode som kan benyttes til at synthetisere menneskelignende gang-mønstre. Dette udføres gennem en bottom-up model af det menneskelige bevægeapparat, som optimieres i forhold til abstrakte formålsfunktioner der f.eks. søger at minimere energiforbrug per strækning. Emnet Predictive Simulation bliver forklaret ved a dykke ned i dets centrale under-systemer Den mekaniske model, det prædikative kontrol system, Simulatoren, Optimerings systemet og formålsfunktionen. Det frit tilgængelige software *PredictiveSim* af Dorn et al. (2015) danner grundlaget for dette studie, både gennem en serie eksperimenter men også som en grundlæggende platform til videre udvidelse. *PredictiveSim* er et eksisterende og funktionelt system der indeholder alle de værktøjer der skal bruges for at opnå State of the Art menneskelignende gang-mønstre. Softwaren bygger på et refleks-baseret parametrisk kontrol system med 77 design parametre, som forbinder sensorisk indput til muskel aktivering. Når parametrene optimeres tillader de system at at udføre continuær autonom gang med indbygget stabilitet. Softwaren bliver i første omgang undersøgt og efterfølgende udvidet for at øge dens funktionalitet. De nye funktioner inkluderer at optimere med forstyrrelser, at styre refleks-parametrene med signaler fra hjernen samt at benytte en Blind Random Search som alternativ optimerings algoritme. Eksperimenterne bliver udført gennem en serie optimeringer og simuleringer med en række a varierende hastigheder og forstyrrelser. De resulterende design parametre bliver da analyseret vha. Machine Learning og Principal Component Analyse, for at undersøge potentielle underliggende mønstre. Flere tendenser bliver fundet, hvilket antyder at systemer som disse, som normalt kræver meget computer kraft for at finde variablene, potentielt kan drage fordel af at benytte Data mining teknikker til at approksimere eller forudsige løsninger.

– This page intentionally left almost blank –

Word List

- Ascending Signals: Signals ascending down the spinal column, originating higher up.
- Supraspinal: Signals originating above the spinal column
- In-Vitro: Test-tube experiments
- In-Vivo: Within the living. Experiments performed on live beings
- Stimulation: Signal that stimulates eg. a muscle to activate
- Excitation: Signal that makes a neural response easier
- Inhibition: Signal that makes a neural response harder
- Feedforward: Signals originating from a centralized controller that is directing
- Feedback
- CPG: Central Pattern Generators
- GRF: Ground Reaction Force
- Proprioception: Feature/Center of the brain that handles relative position of all limbs to each other
- PCA: Principal Component Analysis, machine learning algorithm
- EVR: Explained Variance Ratio, how much of the overall variance that one particular PCA axis can account for.
- COM: Center of Mass

– This page intentionally left almost blank –

Contents

– This page intentionally left almost blank –

Chapter 1

Introduction

Gaining insight into the inner workings of the human body is a long standing goal of medical research. Knowing what is going on underneath the hood enables us to diagnose, treat and get inspired by the complicated mechanism that nature has created and optimized through millions of years. Studies of the locomotor through biomechanics is already widely used to treat disorders and disabilities as well as to guide surgeons before operations and estimate the outcome.

Human gait consist of a complex pattern of muscle stimuli orchestrated by the nervous system, the nature of which is difficult to measure experimentally due to the high amount of signals that need to be recorded simultaneously. Gait studies in biomechanics rely heavily on clinical data of kinematic measurements as well as muscle activation by electromyography (EMG) and to some extent tendon force measurements, when possible. However, many limitations prevail which to a large part is based on lack of sufficient data and understanding of the underlying circuitry of the nervous system. When exploring the circuitry and functions of the nervous system, acquiring clinical data often requires invasive methods that in many cases are not suitable for human trials (Miller and Wilson, 2008; Kralik et al., 2001). Furthermore, the motivation behind movements have been shown to rely not only on physiology, but also on social and cultural aspects as well as mood and other parameters that are difficult to estimate. For this reason qualitative research regarding questions on the motivation behind movements are limited by the variance caused by these unknown parameters.

As an alternative to measuring nerve-signals directly, the muscles serve as a natural amplifier of their motor neurons. Using EMG recordings these signals can in many cases be acquired in a non-invasive way. However, this leaves a big black box for the logic behind the stimuli when only the results can be measured satisfyingly. To gain an insight into this black box the method of Predictive Simulation can be used (See Chapter 2). Predictive Simulation in biomechanics is an opportunity that has appeared with the recent advances in computing power and sophisticated simulation software. Here the structure of the nervous system is explored by a bottom-up quantitative method. Predictive Simulation consists of several sub-systems of which the foundation is based upon a so called Predictive Controller. For gait specific biologically-inspired Predictive Simulation, the Predictive Controller can then be thought of as a hypothetical model of the human nervous system. This controller provides a circuitry that can be tuned by performance optimization on a finite sample set, which then enables it to act accordingly on future datasets. In this way it 'predicts' what action to take based on

on data that it has never seen before. The human nervous system can in theory be expected to possess similar features of a complex predictive controller given that actions and reactions are assumed to be extrapolated from previous experience, thus making informed guesses as to what a feasible response to previously unknown inputs could be. Based on this a Predictive Controller is sought which can predict human behaviour. The controller is tested by performing a Predictive Simulation which results in motion data for a model representing the human locomotor. The motion of this model can then be compared to clinical data for validation of the controller.

Current Predictive Controllers are able to achieve gait-like patterns essentially by optimizing the system to move at a certain speed using the least amount of effort possible. Using speeds within normal human walking (0.8 to 1.8m/s) current systems automatically settles on a movement strategy resembling human walking. Current state-of-the-art of such controllers are able to synthesize patterns that resemble the overall structure of human walking on a flat ground, approaching one standard deviation of clinical data. Using these systems allow access to a well-defined gait pattern that is synthesized from a bottom-up mathematical model where all parameters are known, without the need of recorded kinematics. This gives the model no bias beyond what is included in the components and makes each simulation repeatable, allowing for investigations that are otherwise impossible in human trials. However, there are many obstacles yet before a system is able to correctly predict human motion on a wider scale. This pose a large limitation to current systems. For this reason improving on current Predictive Controllers could prove useful in many ways including what will happen when abnormal parameters are present, pre-exploration of hypotheses before expensive clinical trials are required as well suggesting potential features that are included in a validated model, but haven't been found in humans yet. In this way the benefits goes both ways as a Predictive Simulation system may both be used as a medical tool but simultaneously allow a method for exploring the circuitry of the nervous system theoretically, by inducing its functions by abstract data instead of measuring directly. Entering the topic was found to have a steep learning curve as well as requiring a large amount of time and effort to get started. No overall introduction to the topic of Predictive Simulation was found in the literature.

This thesis therefore aims to provide a overview by giving a introduction of the basic building blocks, both in terms of fundamental literature, theories, techniques and software. However, due to a massive topic relying on theories from several disciplines, this thesis will only provide a brief overview as well as some select details. While an abundance of complex equations are present underneath the hood, this will not be documented mathematically but instead explained in general terms. The present study relies an open source and freely available Prediction Simulation software package called *PredictiveSim* by Dorn et al. (2015). This software revolves around a biologically inspired parametric system that through optimization of high-level objectives is capable of discovering locomotion that partly resembles human gait. The fact that it is Open Source poses another benefit where every edit and progression may serve as an additional steppingstone for researchers to improve rather than recreate.

1.1 Project Objective

Through this thesis three main objectives will be explored regarding the use of Predictive Simulation for biomechanical analysis of human gait. These objectives are:

1. Provide a brief overview of the theoretical foundation needed for approaching Predictive Simulation
2. Establish an accessible and documented basic toolbox for performing Predictive Simulation of Human Gait
3. Show and analyse initial results obtained from gait synthesis using the *PredictiveSim* software

1.2 Approach

The present study was approached by an initial literature study. This allowed narrowing down on the area of designing and performing Predictive Simulation. Through this process *PredictiveSim* was discovered. This is a freely available software package combining all elements needed for producing a state-of-the-art plausible simulated gait that resembles the overall characteristics of human walking. The software is made by Tim Dorn and Jack Wang et al. (Dorn et al., 2015) and published at Stanford University (Further details in Chapter 2). This software then formed the foundation of the rest of the study. First the software was explored, documented and tested while a number of extra features were added. Simultaneously an additional extensive literature study was made specifically on Predictive Simulation. Finally a number of experiments were conducted using the software and the resulting data was analysed.

1.3 Thesis Structure

This theses first introduces the topic in question in this present chapter. Then the general theory behind Predictive Simulation is presented in Chapter 2 followed by an introduction to the Nervous System of the Human Locomotor in regards to designing Predictive Controllers in Chapter 3. The *PredictiveSim* software is then described in Chapter 4 including new features as added during the present study. Chapter 5 presents a consecutive list of experiments grouping motivation, method, results and summary together for each experiment performed. A few Further Thoughts are presented in Chapter 6 and a conclusion is made in Chapter 7 along with a suggestion for future studies.

The Appendix covers a documentation of the resulting extended software as well as some supplementary sections providing additional details on intermediate steps.

Chapter 2

Predictive Simulation

This chapter introduces the fundamental principles of Predictive Simulation followed by an in depth explanation of its sub-systems and how they interact. Finally synthesis of gait is explained by combining the sub-systems and performing a Predictive Simulation.

2.1 The Combined Process

Predictive Simulation is a method that relies on the interplay between the following sub-systems: Mechanical Model, Predictive Controller, Physics-Simulator, Objective Function and Optimization Algorithm. These can be related to their human counterparts as follow (A graphical representation can be seen in Fig. 2.1.

- Mechanical Model: Human Locomotor System, including skeleton and muscles etc..
- Predictive Controller: Nervous System, describing the complex relations between sensory input and muscle stimuli.
- Physics-Simulator: The surrounding world, enforcing the laws of physics and dictates the motion outcome of muscle forces.
- Objective Function: Brain, the area that describes an aim and evaluates the feasibility of a potential manoeuvre in relation to that aim.
- Optimization Algorithm: Evolution and Learning. Targeted curiosity that stimulates the nervous system to perform different manoeuvres that can then be evaluated by the brain and 'learned' if they were successful.

Within the process of Predictive Simulation the Predictive Controller has a key-role by describing actions and performing them through muscle stimuli. These stimuli then result in forces applied to the Mechanical Model. The Simulator evaluates these forces by calculating the resulting motion within the laws of physics and the Objective Function assess its performance. Finally the Optimization Algorithm teaches the Predictive Controller to stimulate better, as defined by the Objective Function.

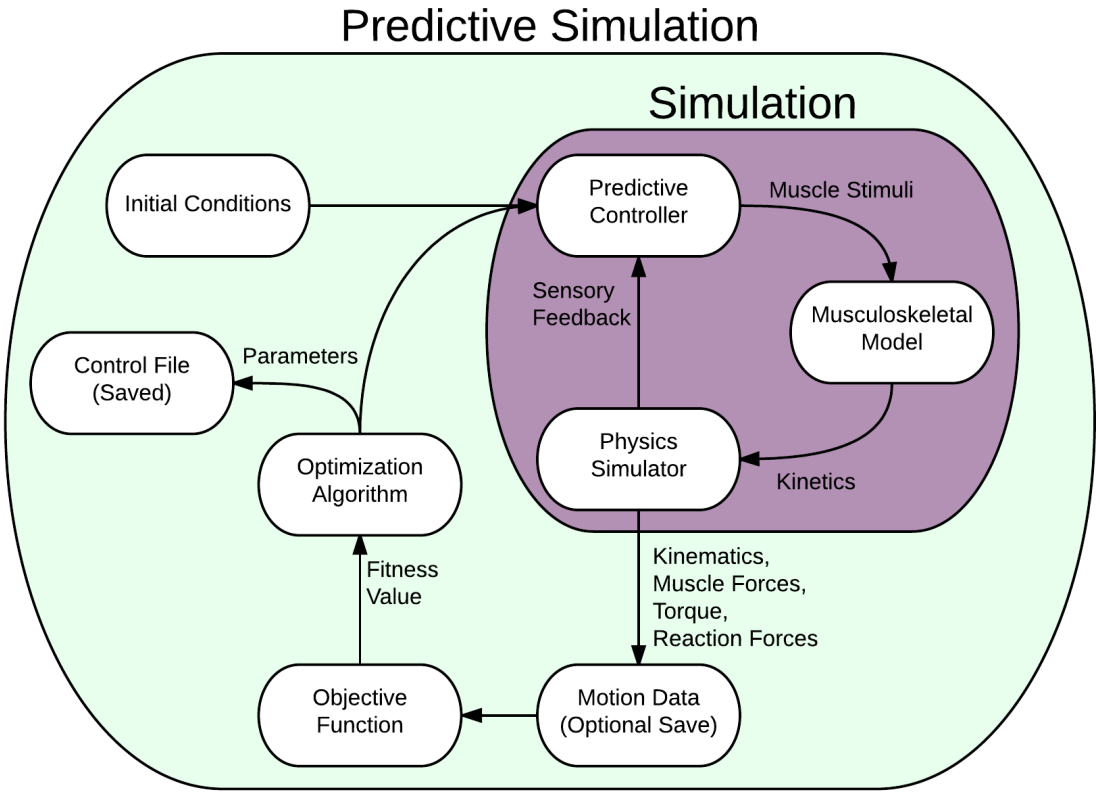

Figure 2.1: HPredictive Simulation Process Overview

2.2 The Mechanical Model

The mechanical parts of the Human Locomotor can be viewed as a multibody dynamical system with a large amount of linear actuators. This system consists of 244 Degrees of Freedom (DOF) with approximately 230 joints. To actuate these there are more than 650 skeletal muscles whereas at least 70 are involved directly in gait. A mathematical representation of this interplay of skeleton and muscles is called a musculoskeletal model. This model consists of a number of sub-models describing limbs, joints and muscle as well as their anthropometric data such as mass, center of mass and inertia. The model can be as simple or as complicated as deemed necessary and may contain any combination of known parameters. This includes varying moment-arm for each joint as well as size, strength, activation and contraction dynamics of each muscle, soft-tissue approximations etc. Due to computational cost and time required to establish good models the musculoskeletal model is generally optimized, simplified and specialized for the question at hand, applying mathematical models that approximate the human counterparts within an allowable margin of error. This includes lumping of sub-models ie. Lowering DOF by combining head, arms and spine into a single rigid trunk, removing DOF's that theoretically have little impact on the question at hand and combining muscle groups, tendons or a range of muscle with similar functionality into single muscle-tendon units (See 2.2.1). For 2D motion it

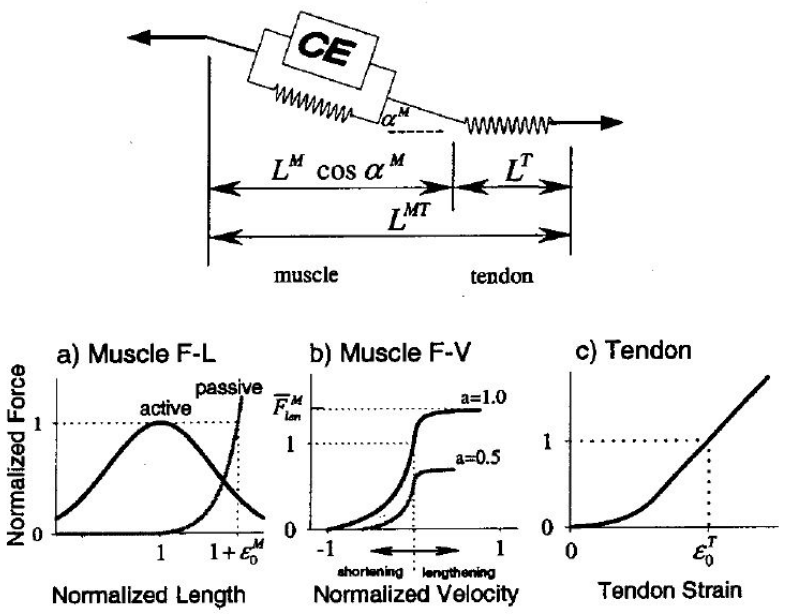

Figure 2.2: Hill-type MTU schematic (Thelen et al., 2003)

has been found that the core functionality of gait can be described by as little as 9 DOF's and 16 muscles (Geyer and Herr, 2010).

2.2.1 Muscle-Tendon-Unit

The dynamics of muscles and tendons are normally combined in a Muscle-Tendon-Unit (MTU). The MTU may vary greatly in complexity. This is in part due to complex activation and contraction dynamics that human muscles posses. There exist many of such models with varying complexity, features and dynamics. A common model is the Hill-Type MTU (Hill and Se, 1938) also known as the 3-element muscle (Fig 2.2). This is made by a Contractile Element (CE), a parallel elastic element and a serial elastic element. These are parameterized among other by the Muscle-Tendon Length (L^{MT}), the Tendon Length (L^T), the Muscle Pennation angle (α^M) as well as muscle activity (a) which affects the string force provided by the Contractile Element (Fig. 2.2) Inspired by physiological muscles, the CE includes activation dynamics that simulate muscle fibre recruitment and relaxation which is shown on Fig.2.3. This describes the time-dependent development of force as a result of an excitation (step response). The CE furthermore contains contraction dynamics which simulates the fact that physiological muscle forces are sensitive to contraction velocity, both in direction and amplitude (See Fig 2.2 for Muscle Force-Length, Force-Velocity and Tendon dynamics). Further correlation with biological systems can be achieved by including aspects such as tendon dynamics, however this may cause a large slowdown as well (Millard et al., 2013). The varying properties of different muscles (as seen on Fig 2.3) can then be specified by parameters based on clinical data.

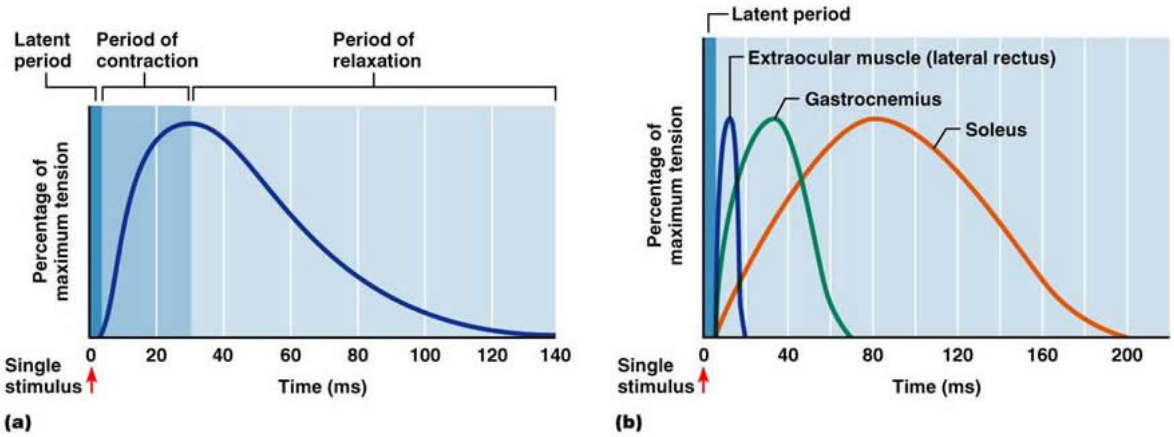

Figure 2.3: Hill-type MTU Twitch Response (Could not determine copyright holder)

2.2.2 Joints and Moment Arms

All rotational joints actuated by linear actuators will include some sort of varying moment arm which can generally be modelled by a sin() function. In the human body this moment arm is further complicated by interaction and collision with other muscles, bones and soft tissues. This results in uneven variations that are difficult to model and linearise. Similar to the MTUs, the varying moment arms are usually only included if the difference is deemed large enough to have a significant impact on the results.

2.3 Predictive Controller

The way Predictive Controllers replicate the functioning of human Nervous System is by describing the inner circuit that relates sensory input to muscle excitation. This includes describing: Each sensory organ by how they react to the surroundings and what information they give, all neural pathways that such information travels as well as the logical or neural decision structure that correlates input to output (See Section 3.6 for a brief overview of building block that are usually used). To approach the creation of a feasible model, that is both functional and simplified enough to be possible, a large range of techniques, theories and data are used. These arise from a range of rather independent disciplines each providing specialized insight into particular aspects including: computational neuroscience for understanding and implementing neural networks, biomechanics for specializing in the overall mechanics of the human body, mechanics for a general understanding of the dynamical systems. In addition Robotics contribute by to an artificial motion planning, AI and Control Theory. Physiology and Biology both give insight into plausible features of the human body and finally computer science provides a means to implement software-related issues and perform physics-based simulations that can yield explanatory results.

2.4 Simulation

The Simulator has the task of solving the Equation of Motion of the Mechanical Model. During the simulation process the Simulator progresses through a series of small time steps. At each

step it evaluates the motion of the Mechanical Model according to both muscle forces and physics while sending the current state back to the Predictive Controller, allowing it to react accordingly for the next step. Eventually the result of this operation is motion of the Mechanical Model.

2.5 Objective Function and Performance Quantification

The Objective Function (Alternatively the Cost- Error or Fitness Function) is established to evaluate and quantify the performance of a model by yielding a scalar value (From now on referred to as *Fitness*). This express a relative number that can be compared to other evaluations. For biologically inspired Predictive Simulation gait models, the Objective Function often seeks to describe the assumed inner priorities of the central nervous system (As explained in section 3.2). This is done in the hopes that the Predictive Simulation system can be made to make the same decisions for motion planning as that of the human body. To facilitate finding a solution within a large solution space, it is crucial to formulate the Objective properly. While maybe only a very limited sufficient solutions exist, this function should help to show the way by defining terms that gradually leads the solution toward the optimum.

Examples are: If the aim of a simulation is for a person to stand up from a couch and move through an open door, the objective can be written i several ways. One is by simply defining success as having passed through the door at the end of the simulation and failing otherwise. This gives no indication of how close a potential solution is to solving the problem, leaving the optimizer to guess blindly for each new sample and making a flat fitness function. If the objective instead was made to give a progressively improved fitness the closer to the door it finishes, the optimizer would know whether it improved or not, compared to previous samples. This gives a concave objective function where the optimizer basically rolls down hill to find the minimum, which is the optimum solution for minimization problems. If local minima exist these form holes in the surface where the optimizer can get caught before having reached the actual minimum.

2.6 Optimization

To find the best combination of all parameters an optimization algorithm can be used (Explained further in 2.6.2). In some cases problems may be approached analytically where the optimum solution can be calculated directly. This generally concern linear systems. However, for dynamical gait using biologically-based actuators, the system often turns non-linear where only the simplest of cases can be solved analytically. Instead an iterative numerical approach may be used. An optimization example is illustrated on Fig. 2.4 where: Some predefined initial parameters are simulated to form a motion. This motion is then evaluated by the Objective Function to establish a measure for performance. A new set of parameters are then defined by an algorithm in the Optimizer and a new simulation is made and evaluated by the Objective Function. The new fitness is then compared to all previous simulations and the parameter set with the best fitness is saved. If the fitness is not sufficient yet the process continues.

This loop can then continue generating new parameter values and evaluating them against the previous until the achieved performance is sufficient.

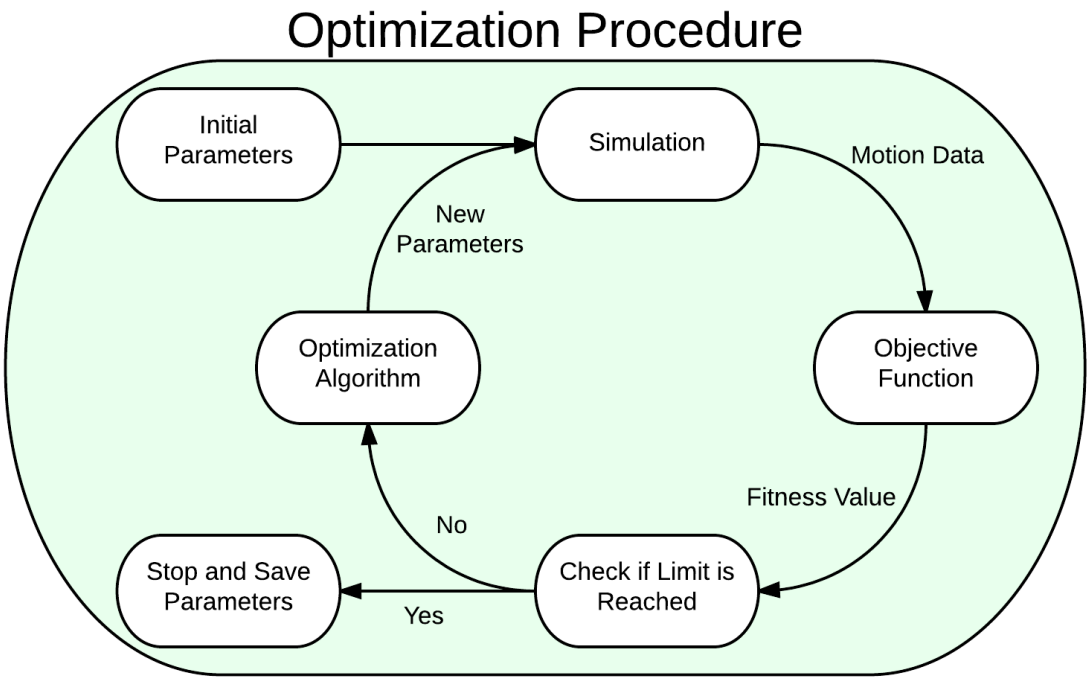

Figure 2.4: Diagram of Optimization Procedure

2.6.1 Performance vs. Bias

Due among other to the dynamics of the Hill muscle, the underactuated aspects of human gait as well as several others, the system often ends up being highly non-linear and unapproachable by analytic methods. Solving biomechanical problems also often ends up with a high-dimensional solution space. This all causes computation time to rise drastically which often is a limiting factor. For this reason it may be infeasible to include everything in a single model, thus simplifications has to be made. Due to the complex nature of the human body and the lack of validated models, it is, however, difficult to know which aspects can be assumed to be driving or driven of the question at hand, thus making the simplification a difficult task by itself. Wrong assumptions and simplifications easily introduce bias in the system, which may give rise to unknown errors or wrong result. Ideally only the aspects backed by clinical data or validated theories should be specified. However, gaps in between established theories or data may need to be modelled anyway, to reach a functional system. By using a method such as Predictive Simulation these gaps are handled by parametric models seeking to explain most likely connections by different combinations of these parameters. This allow a single model to represent a large range of functions by simply varying the parameters. An approximation of the 'right' values can then be found through optimization. In theory almost everything can be described by a large enough list of parameters. However, in reality this is rarely feasible as it would leave too many parameters to optimize. This greatly slows down any iterative process, due to the solution space expanding exponentially by number of parameters. Therefore assumptions have to be made to simplify and include as few parameters as possible.

2.6.2 Optimization Algorithms

There exist a long list of optimization algorithms with a wide range of properties, benefits and disadvantages. A large part of their differences rely on how many assumptions are made about the type and shape of the Objective Function, ie: Can the optimum be assumed within specific bounds, are there many or few local minima, how good do I need to solution to be etc. By making such assumptions the solution space can be narrowed down, allowing a more focused search. A simple stochastic optimization algorithm can be made using a so called Blind Random Search (BRS). This provides a parameter set by generating a random value for each parameter using a uniform distribution between a set of predefined bounds. This set can then be evaluated by the Objective Function to find the corresponding fitness and more sets can be made. The more sets generated the higher the chance of finding a good fitness. However, evaluation may be computation-heavy which is usually the case when simulation is required. This limits the amount of sets that can be evaluated within a given time frame. A search algorithm like the BRS relies on no other preconceptions about the shape and type of the Objective Function apart from being limited within the parameter value bounds. However, if these are wide or the problem has a high dimensionality, the possible combinations expands exponentially and the BRS algorithm may require far too many sets for finding a suitable solution. In this case a more specialized algorithm can be used, which in many cases uses a bias for narrowing in on where it 'thinks' that the best fitness may be. This can be achieved in many ways with pros and cons for each method but if designed well they can show an immense performance improvement compared to the BRS. However, common to all is that the bias may also cause the algorithm to get caught in a local minimum with little chance of knowing whether a better fitness can be found somewhere completely different. The algorithm may not be able to find the global optimum. This issue introduces a sensitivity to initial conditions where many biased search algorithms tend to search for solutions around their initial values. In this way the solution found by the optimizer may therefore change depending on initial conditions, which means that the solutions are not the global optimum. If the Objective Function cannot be solved analytically, it is rarely possible to determine if a point is the global optimum or not. As long as different initial conditions yield different solutions, it is impossible to determine whether the global optimum has been found.

2.7 Synthesis of Gait

The current State-of-the-Art Predictive Simulation systems can produce gait patterns that resembles the basic patterns of human gait without relying on recorded data (Dorn et al., 2015; Dzeladini et al., 2014; Geijtenbeek et al., 2013; Song and Geyer, 2015; Van der Noot et al., 2015). This conclude in a movement that is only biased by design of the system. In this way the motion is well-determined and can be replicated over and over again, contrary to most In Vivo experiments. The synthesis is achieved by combining the mechanical model with the Predictive Controller which is then optimized based on an objective function that describe the proposed inner priorities (See Section 3.2 on inner priorities). For gait studies these priorities usually include a target velocity, an effort term and others such as head stability etc. The optimization then finds the parameters for the controller that result in the best performance. This happens to coincide with the overall motion pattern that all humans exhibit, suggesting that current systems are on the right track. In this way the resulting gait is based on a bottom-up design approach where every aspect of the model is known and chosen carefully, which certifies that

the conceived movement pattern is relying only on the included models. Being able to synthesise gait in this way enables thorough investigations of the underlying mechanisms of gait, where all data in both the mechanical model and the controller can be probed, plotted and handled in whichever way needed. Adjustments can then be made and new optimizations performed, allowing to compare systems and their results with a well-defined difference.

Chapter 3

The Locomotor Nervous System

This chapter introduces a brief theoretical foundation for biologically-inspired Predictive Controllers for gait. The key of designing such controllers is in replicating the human locomotor as far as possible. For this reason much of the theoretical foundation concerns biological insight.

This chapter first gives an overview of biologically inspired gait controller, then explains for categories which relate to such controllers: Inner Priorities, Wiring and Logic, Muscle Reflexes and Supraspinal Control. Finally these are combined into a set of biologically inspired building blocks for creating gait controllers based on biologically feasible modules.

3.1 Biologically-Inspired Predictive Human Gait Controllers

The human nervous system is vastly complex and while select functions have been thoroughly researched not many facts are known. It is here hypothesised that in order for the human to be able to do a range of daily routines, a number of features are thought to be present. These include the ability to:

1. Considering multiple alternative for any action we are could be taking at any point and deciding on one above the rest.
2. Relating sensory input to a suitable reaction
3. Predicting the outcome of our actions and anticipating sensory input
4. Directing movement based on intentions
5. Orchestrating muscle activation to perform intended movements
6. Adapting to anticipated and unanticipated changes from the environment

Creating a system capable of stable locomotion similar to that of humans is a long standing goal for robotics. It has however proved exceedingly difficult even to tackle just some of the points above. For gait studies that are not required to adapt to its environment, the challenge can be narrowed down to point (5) only. This challenge was approached by Anderson and Pandy (2001) who is believed to have implemented the first biologically inspired gait controller able to generate human gait patterns without relying on recorded motion data. This was done

using dynamic optimization of a high-level objective seeking to minimize metabolic cost of travel. The optimization was applied to muscle activity of a musculoskeletal model with 54 Hill-type MTU's. The results were found to correspond to the primary aspects of human gait data. Their system revolved around direct optimization of muscle activity for each muscle. This was done by letting the optimizer define the amplitude of a series of node-points over the timespan of the simulation. Each node-point then described an activity amplitude at a specific time. This enables approximating patterns seen from EMG measurements of muscle activity. While this system allowed any muscle activity that could be described by the given node-points, the amount of node-points needed was relying on the total timespan of the simulation, which made long simulations infeasible. Furthermore, each optimization would be specific to the exact circumstances. This was due to the fact that the result was a series of amplitudes over time, that could not adapt to changes.

In 2010 Hartmut Geyer and Hugh Herr presented a different model that describe a Predictive Controller capable of generating stable gait patterns by means of describing the logic behind the excitations rather than the excitations themselves. This was done by a reflex-model reacting to sensory stimulus from feet, muscles and posture and relating them to muscle excitations (Geyer and Herr, 2010). By defining a set of rules on how each muscle should react to a combination of sensory stimuli and tuning a range of parameters to match a certain speed, they were able to generate a continuous and stable gait pattern with inherent balance, which was found "to predict some individual muscle activation patterns known from walking experiments". In this way Geyer and Herr described a proposed nervous system for their model, as well as trialled it against human data. This system enabled additional features from the list above including "Relating sensory input to a suitable reaction" and to some extent "Adapting to anticipated and unanticipated changes from the environment". Their publication has paved the way for several other studies including Wang et al. (2012) who extended the controller to automatically realize a running motion when optimized for higher speeds; Geijtenbeek et al. (2013) who generalized the walking controller for different morphologies while improving stability to cope with uneven terrain and external perturbations; Dzeladini et al. (2014) who implemented Central Pattern Generators (CPG) for performing speed modulations, thus including a means for "Directing movement based on intentions"; Dorn et al. (2015) who explored walking on inclines while carrying varying loads; Song and Geyer (2012) who extended the controller by adding a feature for sending varying parameters to the reflex-controller during simulation, thus enabling speed changes;Song and Geyer (2015) who extended this further by added even more capabilities for "Directing movement based on intention" enabling turning, walking, running, walking on stairs, accelerating and deceleration etc. All of the mentioned studies have furthermore explored different biologically-based hypothesis' that through experiments have been shown to either improve or decrease correlation with human data. CPG's as used by Dzeladini touch upon a paradigm parallel to that of the reflex-based model, where gait patterns are believe to stem from groups of neurons in the spinal cord, that generate continuous patterns which can be modulated. While CPG's have been found in other mammals and proved to be capable of generating gait patterns without connection to the brain, these have yet to be observed in humans. This thesis makes no assumptions as to which, if either, of the two paradigms are correct. However, the present study is based upon research that derive from the reflex-based controller model proposed by Geyer and Herr (2010) using Dorn et al. (2015) as the starting point.

The model proposed by Geyer and Herr (2010) is based on some physiological principles as well as a functional analysis of legged dynamics. In this way the capabilities of the nervous

system were evaluated and sought recreated by any means possible, where biologically feasible methods were preferred rather than required. Their model is based on a controller that is entirely feedback based, which corresponds to physiological reflex loops. However, nu literature currently supports that humans should be able to walk on reflexes alone (See Section 3.3). In this way all muscle activations are a direct result of feedback from one or more sensory organs, relying only on muscle activation dynamics and neural transmission delay for timing. The sensors used include foot contacts (mechanoreceptors), internal muscle sensors (Golgi and Spindle), as well as certain arbitrary joint and limb angles, which can be thought of as vestibular and proprioceptive senses. These sensors are all connected to muscles by a range of logical circuits, Proportional (P), and Proportional-Derivative (PD) controllers that either excite or inhibit different muscles (see fig 3.1). Each circuit is parametrized by values such as gain, K_p and K_i as well as target values for the P and PD controllers. The actual values of these are unknown but the controller is designed based on the hypothesis that there exist a combination of these parameters that can result in coordinating a gait pattern autonomously.

Designing such controllers take a lot of work but can be approached as a way of reverse engineering the muscle activity. This can be seen as a black box where input is all the sensory organs, which are fairly well understood and output is muscle activity. The job is then to connect these using biologically plausible methods in order to to replicate muscle activation dependent on estimated input sensory information (See Building Block in Section 3.6). I.e. If the design were to be founded on clinical data a sample could be recorded with muscle activations and length variations as well as corresponding reaction forces from the ground. These would then be sought correlated by means of biological building blocks (See Section 3.6). Each building block possess a different response to input. If, for example, consistent patterns are found where one muscle activation is found to increase along with another muscle being rapidly lengthened then these could be related by a positive Length-Feedback. If other relations are found that appear consistent during only certain phases of movement, then 'states' can be added that can change behaviour of such building blocks. For stance and swing phases such a state could be implemented by connections relating the activation of weight-carrying muscles in each leg that then serve as reciprocal inhibition of other connections and consequently changing the gain of of some connections based on how much each leg is currently carrying (The whole body weight during stance and leg-weight during swing). A similar system is included in the controller by Song and Geyer (2015). Each of these building blocks now forming a controller are then tuned by some parameters. Finding the combination of these values that give the best walk pattern is then the primary aim for optimization of gait controllers.

In Geyer and Herr (2010) 2 states are furthermore specified, separate for each leg. These allow the system to trigger different sets of parameters depending on whether each leg is currently in it's *swing* phase or *stance* phase. The states are triggered by whether or not foot contacts are touching the ground. In this way the overall circuitry of the model is within reasonable proximity of what is known from physiology, where a few liberties are taken that currently have little to no foundation in literature or clinical data. However, the model generates gait patterns nonetheless and has established the foundation for others to improve upon.

3.2 Inner Priorities

Realizing why the human locomotor moves as it does is a key to understanding how it work. The particular movements of a human can be thought of as the product of the capabilities of the

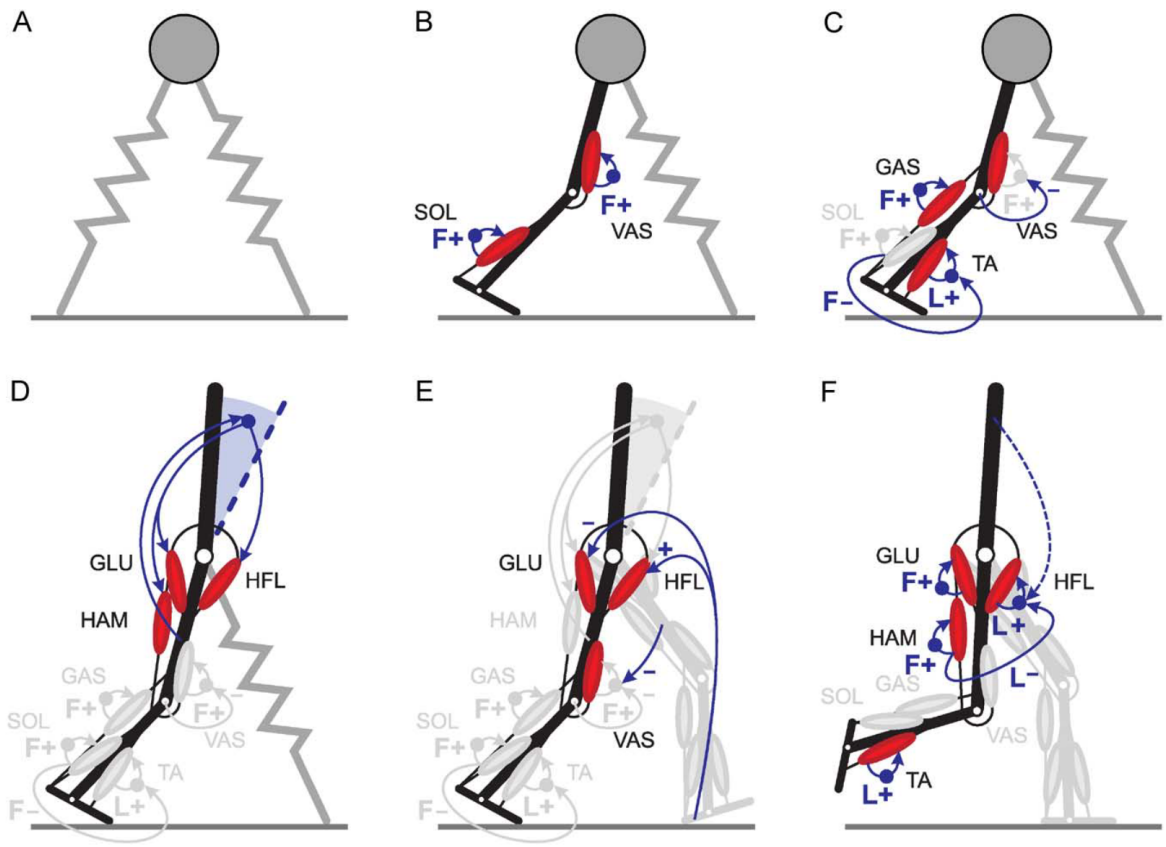

Figure 3.1: Geyer and Herr Reflex Controller (Geyer and Herr, 2010). F+ and F- describes positive and negative Force Feedback where L+ and L- describe positive and negative Length-Feedback. Arrows show where the input originates and which muscles the output stimulates. The torso angle is controlled by a PD-controller which stimulates multiple muscles simultaneously.

body as well as a constant prioritizing that choose certain actions over other. From a robotics point of view, this problem if often described as optimal control, which all-things-considered provides the best solution to the task at hand. While humans are not logical machines and may occasionally choose sub-optimal solutions, the optimal control problem is believed to be a good estimate statistically (Künzell et al., 2013). To approach the question of why humans move as they do, Predictive Controllers are used as a means to test different rules or objectives by simulation. While it is unlikely that a single objective can serve as the definition of optimal control for all tasks, looking for what may be of most importance is a good starting point. In biomechanics several of such objectives have been explored already. The most popular objectives concern the effort required to perform a certain task. Examples include:

- minimization of the sum of squared torque around all joints
- minimization of sum of squared muscle activity
- minimization of metabolic cost per length travelled
- minimization of muscle fatigue

These objectives all express the notion that the human body is believed to move using the least amount of effort, thus saving energy for other actions. As is also seen, effort can here be expressed in several different ways which have all been found to result in different movement strategies. A comparing review was made by Ackermann and van den Bogert (2010), which concluded that muscle fatigue is the one generating the muscle activity that most resembles humans. This suggests that the human decision process value endurance over long term energy consumption. More generally these objective serve to explain how a problem with many solutions should be solved best. These types of objectives are often referred to as high-level when the actual problems are only arbitrarily related to the objective. I.E. where minimizing effort of the body causes abundant collateral effects on a large range of sub-part. High-Level objectives are used among other for solving the muscle redundancy problem arising from the fact that each joint is actuated by several muscles that may be combined differently to achieve the same torque. Objectives like these arise from informed guessing as well as evolutionary and medical observations. The mentioned effort minimization may, however, only be a part of it. By looking at gait adaptations for entering on an un-safe surface it is clear that effort may not always be the most important. On ice or any other slippery surface the motion pattern changes drastically and muscles react with a different strategy to accommodate uncertainty (Cham and Redfern, 2002). Explaining this as well may require a revised objective which includes stability in some way. The fact that humans appear to somewhat follow a least-effort objective when not challenged by stability and only changes behaviour when needed could suggest a hierarchical structure. The combination of effort and stability could then be described as a check list, where stability is the first in line and effort second. Explained differently the stability term could be specifying the bounds for the solution space that honour its requirements. These new narrower bounds are then passed on down the list where effort may find the best solution within those narrower bounds. Many other such entries are likely to exist on such a list, including realizing current intention, which may to appear beneath stability and above effort. This would mean that stability is primary, then execution of an intention and finally the action is performed using the least effort possible.

Beyond that it is well known that skeletal muscles co-activate with their antagonist. This has to our knowledge not been replicated successfully in simulation yet, apart from being introduced as an empirical ratio between agonist and antagonist. The nature of co-activation is still up for debate but potential theories include control of joint compliance and stiffness (Annunziata and Schneider, 2012). Describing this logically and implementing it in the objective function could help to investigate the cause of muscle co-activation. Additional objective terms may also include minimization of head-movement during gait, which is induced from empirical data, and many more can be proposed, ie: minimization of pain, accelerations, impact amplitude, wear/tear and tissue forces. The two latter could potentially be thought to contradict the co-activation given that co-activation drastically increases bone-on-bone forces. Based on this it is clear that the objectives of the nervous system is a complicated matter which may include a long list of different constraints and criteria, weighted against each other in one way or another.

3.3 Wiring and Logic

The path and dependencies of gait-specific skeletal muscle innervation is much debated. While the immediate stimulus of lower limbs have been found to originate in the lower spine, the path and information passed through the descending signals as well as the underlying logic

behind them are still a mystery. It is logical to assume that the brain is the major player in most human actions but the questions are to what extent and what are the alternatives.

To differentiate reflex stimuli from everything else, the latter is referred to as supraspinal input, meaning everything descending from above the spinal column. Furthermore, stimuli as a direct response to sensory input is referred to as Feedback where stimuli appearing as a supposed result of intention in the brain and sent out to the muscles is referred to as Feedforward. The model suggested by Geyer and Herr (2010) relies entirely on reflexes to generate the gait pattern.

So far there are no indication in the literature that humans can walk based on reflexes alone and experiments on patients with spinal-cord injury suggest that feedback integration is essential for performing gait (Sinkjaer et al., 2000; Dietz, 2002). However, other mammals such as cats have been shown to be able to walk with a severed spinal cord (Duysens and Van de Crommert HW, 1998), which suggests that similar functionality could theoretically be possible for humans. Dzeladini et al. (2014) suggests that a plausible combination could be that proximal muscles are more directly controlled by the brain than distal muscles, which rely to a higher degree on feedback.

Geyer and Herr (2010) also introduced states in their controller. While it may appear infeasible that a biological system would have actual states that are either on or off, it can be thought of as a reasonable simplification that can explain some of the behaviour of adaptive reflex loops. A more physiologically feasible model appears to be a reciprocal inhibition of multiple reflex-loops (Sinkjaer et al., 2000; Geertsen et al., 2011). Seeing this happening in one part of the body appears to suggest that several logical circuits for each reflex loop can be combined with varying amplitude thus allowing reflex behaviour modulated by external stimulus. This adds an important building block to the collection (See Section 3.6).

3.4 Muscle Reflexes

The particular neurology of skeletal muscles are fairly well understood. Beyond the contractile elements and their supporting constructs, each muscle also contain 2 sensory organs: Muscle Spindle and Golgi Tendon (Proske and Gandevia, 2012). The Spindle reacts to the current muscle length as well as lengthening velocity while the Golgi tendon reacts to tendon force. These two sensory organs provide proprioceptive feedback that is sent both to the brain but also to reflex loops that for lower-limb muscle are located in the lower spine. These reflex loops relate the incoming sensory stimuli to produce a resulting stimuli by a simple logical circuit. For reflexes affecting the same muscle from where the sensory stimuli originates this stimuli is passed on to the muscle itself through the α-neuron (Auto-muscle reflex). Current literature suggest that these auto-muscle reflexes all behave in the same way where sensory input from the Golgi tendon performs inhibitive stimulus on the muscle (Negative force-feedback) and Spindle stimuli perform excitatory stimulus (Positive length- and velocity-feedback). However, Geyer et al. (2003) argue that there is a functional benefit for positive Force-Feedback which could suggest that it should not be left out of consideration. The same sensory stimuli are also thought to affect other muscles where changes in one muscle sends an excitatory or inhibitive stimulus to another muscle. While these connection are difficult to verify experimentally, they provide a beneficial building block for designing a functional controller.

3.5 Supraspinal Control

Supraspinal is the general term for signals originating above the spinal column, including the brain. These signals are generally believed to play an important role in both motor learning and adaptation to task variations where reflex-loops instead account for short-latency high gain responses (Wagner and Smith, 2008). Reflex-loops (Also referred to as Feedback) by themselves provide fast response to perturbations but no ability to change behaviour, ie. stopping or starting to walk, jumping, turning etc.(Wagner and Smith, 2008).

Current systems that combine Feedforward and Feedback for gait controllers include: Dzeladini et al. (2014) who suggests a model where feedforward acts as an observer of the movement, thus enables predicting sensory inputs which allows the model to realize when unexpected perturbations appear; Song and Geyer (2012) who describes a method that implements supraspinal control by varying reflex parameters for achieving gait speed variations; (Song and Geyer, 2015) who developed this approach further where feedforward provides adjustments to reflex gains as well as foot position, which they show as being sufficient for performing stable gait patterns as well as producing a number of different behaviours. These include turning, walking on stairs and speed change; Van der Noot et al. (2015) who combined CPG and reflexes to perform stable gait patterns on a robot, where speed variations were controlled by phase change of the CPG. Mugge et al. (2010); De Vlugt et al. (2002) both suggest that feedback parameters such as force and position are constantly tuned dependent upon experienced perturbations. Forbes et al. (2011) shows that reflex parameter modulation appear to be task-specific. An important role of supraspinal feedforward is also found in impedance variation, where humans have been shown to increase locomotor impedance when exposed to uncertain terrain and lower it again when back on stable ground (Takahashi et al., 2001), ie: when walking from land onto a frozen lake, or when briefly slipping in a soggy fallen leaf the body quickly establishes a recovery strategy after which a more careful gait is performed until the terrain is deemed trustworthy once again. For that reason supraspinal control is believed to be a key feature allowing further investigations in the adaptive control. In order to achieve such impedance adaptation, stability needs to be quantified. Exactly how this is done is still up for debate and is a another topic by itself. Roboticists often use a so called *Zero-Moment Point* (ZMP) that the motion planning algorithm aims for. Other methods include *limit cycle walking* where global stability is considered, allowing for temporary off-balance stages that are later brought back into balance. Geyer and Herr as well as all the controllers in this thesis fall within *limit cycle walking*.

3.6 Physiological Building Blocks

For approaching design of a biology-based Predictive Controller a number of potential building blocks are here defined. These should be seen as a very rough representation of some functions that are believed to exist in biology and provide a fundamental modular structure for generalizing controller design. The list is far from exhaustive and entries such as P/PD-Controllers and States may not be directly based on biology but have been included based on their appearance in the Predictive Controller of the present study

- Senses
  - Mechanoreceptors (Touch)

  - Golgi Tendon (Muscle Tension)
  - Spindle (Muscle Position and Speed)
  - Proprioception (Relative Location of Limbs)
  - Vestibular System (Orientation)
- Logic
  - Neuron (Firing Threshold)
  - Constant Value
  - Sum (Neuron firing)
  - Gain (*)
  - Inhibitory (-)
  - Excitatory (+)
  - P-Control
  - PD-Control
  - Neural Transmission Delay
- States (Reciprocal Inhibition)

Using these building blocks several control laws can be made, Ie. Positive Force-Feedback is a Golgi Tendon connected to a gain with excitatory output, PD-Controller can receive K_p and K_d from other control laws or define them as Constant Value, an angular control signal can come from Proprioception or the Vestibular system and the target can be defined similarly with gains inserted appropriately.

3.7 Discussion on Biologically-Inspired Predictive Simulation

One problem common to all optimization methods that seek to replicate life is that only the question at hand is evaluated. For this reason the solution found will at best ever be the optimal solution for that particular problem. A different solution could be required when asking the same model to solve a different problem. While humans definitely adapt to the task at hand, the underlying system is likely not to change. If the structure of this system is the aim of the optimization, the solution should therefore in theory not change when given a different task. This can be described by evolutionary observations where the system is expressing the best solution to the overall requirements given by the task and all current and previous experiences. Evolution has been found to make compromises where the requirement of performing different tasks are incompatible. This is done while maintaining ideal solutions for both. It is here assumed that the evolution of the locomotor system strives for optimizing energy efficiency, survival and diversity in capabilities.

If instead the optimization is only asked to find the optimal solution for gait on even ground, the outcome may be a strategy that is actually better than humans, thus being super-optimized so to speak. This happens as a result of the model being allowed to specialize in a single task instead of being required to do everything a human needs to be able to do. This issue is to our knowledge not possible to solve completely, but suggests the importance of performing optimization in a rich environment over a broad range of tasks in order to achieve the most human-like behaviour. Some have approached the issue by consecutive optimization of a list of tasks and objectives (De Lasa and Hertzmann, 2009)

Chapter 4

Software Implementation

PredictiveSim is a free and open source software system for performing Predictive Simulation of human gait. The system was released in April 2015 by Tim Dorn and Jack Wang et al. from Stanford University. This was along with their publication *"Predictive Simulation Generates Human Adaptations during Loaded and Inclined Walking"* (Dorn et al., 2015). *PredictiveSim* was chosen as the foundation of this project as it contains all the fundamental tools for performing predictive simulation. This section provides an overview of the software. Further details on specific areas are given. Finally a list of additional features are implemented and presented. For an overall documentation see Appendix A.

4.1 *PredictiveSim*: Software Presentation

PredictiveSim was developed as a tool for investigation of a single controller capable of achieving good performance when carrying loads and walking on inclines. Using this system Dorn et al. were able to generate stable unloaded walking where "joint angles and moments were generally in agreement with human walking" (Dorn et al., 2015). The system is based on (Geyer and Herr, 2010) using similar reflex-based controller with some differences that allow walking on inclines and simulating varying backpack loads. Similar to the system presented in Chapter 2. Five sub-systems are present: Model, Controller, Optimizer, Simulator and Objective Function, (See fig 4.1 for overview). The system is a software package written in C++ and makes extensive use of both SimBody (Sherman et al., 2011) and OpenSim (Delp et al., 2007) through their API. The former which is a multi-body dynamics engine and the latter an extension that adds tools specifically for biomechanical analysis. Simbody is used as its Simulator and the remaining sub-systems will be explained in Section 4.1.1 through 4.1.4. A summary of features, input and output files are given in Section 4.1.5.

4.1.1 Model: Musculoskeletal Model

The Musculoskeletal model used in this system is defined in the *osim* format (OpenSim model description format (Delp et al., 2007)) and is limited to 2D movements in the sagital plane. The model is set to a height of 1.88m and a mass of 80kg. It consists of 7 limbs, 9 DOF, 4 foot contacts and 16 MTU's The limbs are: Pelvis and left/right of Thigh, Shank and Foot. The model furthermore defines a ground plane as coordinate reference. Each limb is then connected

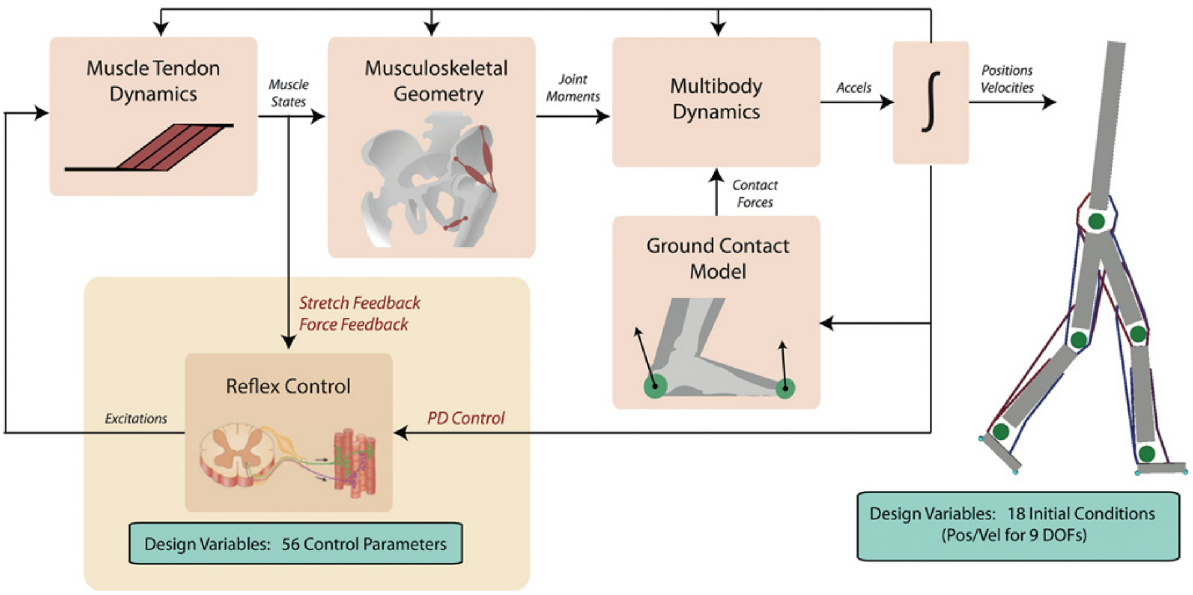

Figure 4.1: PredictiveSim System Diagram (Dorn et al., 2015)

by a planar joint and the Pelvis is able to translate on the x- and y-axis, resulting in 9 DOF (See fig 4.1). The 4 foot contacts are modelled as contact geometry on the heel and toe of each foot, using Hunt-Crossley forces (Hunt and Crossley, 1975) to estimate friction, force magnitude and direction. The 16 MTU's of *PredictiveSim* are based on a custom MTU made by Tim Dorn and Jack Wang which resembles a Hill-type muscle as explained in section 2.2.1.

4.1.2 Controller: Reflex Model

The controller is largely based on (Geyer and Herr, 2010) which explains a purely feedback based system where a list of static parameters and initial conditions generate a walking gait. 3 main feedback control laws are used: Force Feedback, Stretch Feedback and PD Control. Each law is applied multiple times across all muscles, connecting either from one muscle to itself, from one muscle to another muscle or combining input from a list of muscles, aiming for a certain feature angle. Each applied law has a list of free parameters that are found by optimization. The controller furthermore incorporates 3 different states triggered by foot contact. These 3 states describe the phases of stance, swing initialization and stance preparation respectively and switches between different reflex behaviours. Further details and exact equations can be found in (Dorn et al., 2015). The controller relies on a total of 77 parameters whereas 70 are being optimized as default. 18 parameters describe initial condition, 53 concerns the the controller parameters and the last 6 describe the Hunt-Crossley contact force. All parameters are listed in Appendix A.6.

4.1.3 Objective Function

As mentioned in Chapter 2 the Objective Function evaluates a simulation and provides a scalar value that quantifies relative performance. The Objective Function as given by (Dorn et al.,

2015) is defined by:

$$R = w_{fail}J_{fail} + w_{vel}J_{vel} + w_{head}J_{head} + w_{effort}J_{effort}$$

This incorporates penalties for failing, velocity, head movement and effort. Each w specifies a *weight* factor and J specifies a penalty function. Failing is in this case when the COM drops below 0.7m and the penalty is lowered the longer the simulation lasts. This is done to facilitate the sliding scale where the longer it balances the better it is, despite it eventually falling. Velocity is defined as the difference between average velocity and target velocity specified by the user. This term also includes an error threshold of ± 0.05m/s to allow a tolerance where the term gives zero penalty. *Head* accounts for the forward movement of the head relative to the COM, inspired by the apparent tendency for humans to stabilize the head, likely due to maximizing precision of the visuals and stabilizing the vastibular system. Effort is approached by a model that approximates the overall metabolic cost of moving. This includes basal metabolic cost as well as cost of activating individual muscles. This term also includes a penalty for hyperextension which is the sum of squared joint limit torques. This gives penalty if motion is hitting the mechanical limits, probably approximating pain. The specifics can be found in (Dorn et al., 2015).

4.1.4 Optimizer: CMA-ES

The optimization process is initiated by a number of values defined in a Matlab file. These describe the initial values of all parameters as well as which parameters are to be optimized while defining the bounds of each parameter. It was deemed outside the scope of this thesis to analyse further why these exact parameters are used, as they have been shown to work through Dorn et al. (2015).

The Optimizer uses a stochastic optimization algorithm called Covariance-Matrix Adaptation - Evolution Strategy (CMA-ES), presented in Hansen (2006). This algorithm is a derivative-free evolutionary algorithm intended for non-linear and non-convex continuous functions. Furthermore CMA-ES makes only few assumptions about the objective function, using the rank of each objective evaluation as a means to predict where the next likely improvement may occur. The algorithm work on a specified number of parallel processes taking a set of initial values. Multiple lists of parameters are then generated by applying a gauss-distributed perturbation to these values. The perturbation is scaled for each parameter by a covariance matrix (CM) and a step size. The CM estimates a variance based on previous history and and is updated for each iteration. The step size defines the general variance of the perturbation and is progressively decreased based on a dampening factor. The resulting parameters are then evaluated by an objective function yielding a fitness that define the rank for each process compared to the other parallel processes. A set number of "parents" then define the number of highest-ranked parameter sets that are to be kept. The mean values of these parents are then used as the new initial values and the loop starts over. This process runs until it is stopped either by reaching a desired fitness or when a maximum amount of iterations has passed.

This optimizer is then applied to the objective function (See equation 4.1.3) as a minimization problem:

$$min(R(\vec{P}))$$

where $\vec{P}$ is a vector containing all optimization parameters, (in this case 70). During the optimization the algorithm is finding the combination of the 70 parameters yielding the lowest possible fitness value.

4.1.5 Summary on Software Description

When compiled the system offers Optimization, Simulation and Visualization. These are explained below and further details can be found in the documentation, Appendix A.

Optimization: This is performed by specifying among other target velocity, time span of simulation, step size and amount of cores to run the optimization on. Out of these cores (Master) will be dedicated to handling the optimization algorithm, sending out parameters to test on the rest of the cores and keeping track of which solution is the current best. The rest of the cores (Slaves) will then receive parameters, perform simulations, evaluate them and send back the fitness value. Each time the master receives a fitness from the a slave that is better than the previous best, a control file (result_itr[iteration number].sto) is saved out containing the parameters that caused this improved fitness.

Simulation: This is both part of the optimization process but can also be performed individually. This is done by specifying a control file as made by the optimization process as well as a simulation time. This will apply the parameters specified in the control file to the controller and initiate a simulation based on this. The resulting data will be saved out as a simulation file (_results.sto). This file contains a time series of data including joint angles, muscles activation, ground reaction forces, joint torques and many more. Each of these are specified by a value defined for each small time step through the range of the simulation.

Visualization: can be done either during the simulation or as a subsequent action where a simulation file is previewed. This shows a window where the mechanical model is moving according to the simulation data.

4.2 Adding Features

Four new functional features were added to *PredictiveSim*. This was done to enable more flexibility in the ability to find and combine a wider range of parameters. These four features are explained in the following sections. First the ability to save out all simulations to a single file, then perturbations were added as a way to optimize while perturbing the simulation, thirdly Supraspinal Control was added which allow to send commands to the controller during simulation and finally Blind Random Search was added as an alternative optimization algorithm enabling unbiased search through the solution space.

Parallel to this a number of python tools were made that allowed automatic handling, organization and visualizing of data generated by the updated software. Further tools were made to automate repetitive tasks involved in performing batch simulations, monitoring progress and optimizing performance (See Appendix B.2.5).

4.2.1 Save All Simulations

A feature was added that allowed for saving out every single simulation performed during an optimization. This was in contrast to the default behaviour where only simulations that improve upon previous best solutions was saved out. The function was implemented by saving out all parameters and the resulting fitness as singe lines in a CSV file. This feature could then

be enabled by stating "-saveAllSim" as an input argument when launching an optimization. Retaining all samples enabled further statistical methods for analysing the solution space.

4.2.2 Perturbations

Perturbations are often used in motion optimization as a means to quantify and control stability. Controllers optimized without perturbations are likely to exhibit a very low tolerance to disturbances which in most cases is undesirable. Adding perturbations allow the optimization to find an optimal solution while maintaining a margin of stability. For this reason it was deemed important enough for being added as a built in feature, allowing a broader range of experiments to be conducted including stability as a factor. This feature was implemented using an OpenSim *Prescribed Force* (Delp et al., 2007). To do this a new model file was made with an added entry for a Prescribed Force (*Humanoid2d_perturb.osim*). This was set to act horizontally on the torso, 10cm above the COM, with the perturbation force initiating 2 seconds in and lasting for 1 second with instant effect. The amplitude of this force was set to: $F = 50 \cdot pertAmp$ where $pertAmp$ could be specified from the command line using the argument $-pertAmp$ [$amplitude$] (See PredictiveSim Documentation for more information, Appendix A). The addition of perturbation allows to quantify stability by the given amplitude that the system is able to withstand.

4.2.3 Supraspinal Control

Supraspinal control was implemented based on previously mentioned literature in section 3.5. Inspired by Song and Geyer (2012) ascending signals are thought of as an abstract command sent to the lower spine where the reflexes perform the actual orchestration of muscle. This was done by allowing for on-the-fly change of all optimized controller parameters. Given that each set of parameters makes the model move or behave differently, it was hypothesised that the supraspinal high-level control could be a command that simply changes the current parameters or settings in the spinal reflex loops. Ie. using parameters that were found to generate a walk of 1.2m/s and then switching to parameters that were found to generate 1.5m/s. It is speculated that this mechanism could be a general theory for how the nervous system divide tasks into layers where all movements in a sense is feedback based but parameters change continuously based on feedforward signals. This is partly in line with the method applied by Song and Geyer (2012), however their method involved a simplification of parameters, using a fitted curve to interpolate between a few parameters only, that showed a strong correlation with speed changes. The method of the present study applies a whole parameter set. Using a whole set makes sure that no small variation of thought-to-be unimportant parameters get lost in a simplification which could turn out to cause unnecessary errors. Beyond that it includes the ability to use any set directly without prior analysis allowing to use speed and perturbation interchangeably. However, with no logic behind the variation of parameters, each potential state has to be optimized separately to find a suitable parameter set for this. It was theorized that a large amount of optimizations could be performed with N varying parameters such as speed, perturbations and acceleration, mapping out an N-dimensional space where the supraspinal input could be made to pick and choose the parameters best suiting the current intention, considering current state and target state and interpolating values between it's nearest neighbours. Song and Geyer (2012), however, identified the need for specialized transition

settings for variations larger than 0.2m/s. A method for accommodating this was attempted implemented in the present study but was never finished (See Section 4.2.5)

4.2.4 New Feature: Blind Random Search

As described in section 4.1.4 the CMA-ES algorithm is highly dependent upon it's first random sample after which it only searches in the proximity of this. To get a broader look at the solution space of PredictiveSim a Blind Random Search (BRS) optimization algorithm was implemented as an alternative to CMA-ES. This was done by applying a random uniform distribution for each parameter using bounds as specified in the $optimization_bound.sto$ file generated by the Matlab script (See Appendix A). The resulting method is unbiased and searches the whole solution space within the given bounds. Command line initiation of BRS instead of the CMA-ES algorithm was implemented by adding the argument: $-brs$.

4.2.5 Not Fully Implemented

Optimizing for State Transition Based on new needs deriving from supraspinal control, a feature for performing optimization for transitions between states was planned. However, due to technical obstacles and limited time this was never successfully implemented. The feature involved a method for saving out a full system state of a simulation at any given time. This state would include all information needed for recreating the same system state in a new optimization. Along with a feature for loading such state as the initial state of an optimization it was hoped that transition parameters could be found. An example could be to save out the system state from an simulation of 1.5m/s at a point after the gait pattern had stabilized. By loading this state into a new optimization aiming for 1.8m/s and only optimizing control parameters (Not initial conditions) would then theoretically find the best solution for transitioning from the initial state into approaching the new target. It would then be an interesting case to see if the system would be able to perform the whole transition and stabilizing speed when reaching target velocity, or that a better approach would be to optimize for positive and negative acceleration which could then be combined for transitions, potentially as an optimization parameter.

Target Acceleration To accommodate optimizing for target acceleration rather than steady speed another feature was planned. Here the Objective Function was extended to include an option for targeting acceleration rather than a speed. The feature was implemented and is potentially working but time did not allow to linger and perform sufficient testing to accept it as a functional feature.

Chapter 5

Experiment Process

A number of experiments were carried our during the course of the present study. Each experiment is presented in this chapter, singularly, according to the following structure: Motivation, Method, Results and Summary. All experiments concern walking, where different combinations of input arguments were optimized as well as repetitions of identical situations. The input arguments in question are: target speed and perturbation.

Each experiment was performed as a series of optimizations on the HPC cluster at DTU. A single optimization is referred to as an optimization where a related group of separate optimizations are referred to as an optimization run. This includes when multiple repeated optimizations are made using same initial conditions (Which are furthermore referred to as *Independent*) or when a series of consecutive optimizations are made, where each new optimization initiates from the previous best solution (Which are referred to as *Consecutive*). Multiple datasets were made during the following experiments. These were each given a label and are reused across several experiments. Each method section introduces their used datasets along with the label, which will subsequently be used for referring to that data. All optimizations used default initial conditions as provided by Dorn et al. (2015) and were stopped after reaching 30 consecutive iterations where no further improvement had happened. Furthermore the model file with added *Prescribed Force* was used: *Humanoid2d_perturb.osim* (See Chapter 4 for details). The *Optimum* refer to the best solution found during an optimization, which is defined by a control file including both its fitness and the parameters that caused it.

5.1 System Performance Analysis

Motivation

To get a feeling of how the software behaves in general a system performance analysis was carried out. This included exploring how an optimization develops, how fast it converges and whether it arrives at the same solution repeatedly.

Method

Used Datasets

- *default5*: Optimizations: 5, cores: 20, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.
- *sampling1d5*: Optimizations: 10, cores: 20, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.
- *sampling1d5-c80*: Optimizations: 10, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.

Optimization Development: Using dataset *default5* the resulting control files for each improved iteration were plotted showing development of parameter values over iterations (Fig. 5.1)

Comparing Solutions: Using datasets *sampling1d5, sampling1d5-c80* the best iteration for all optimizations were plotted showing the varying parameter values for each individual sample on the x-axis. Parameters were filtered to only show those with a standard deviation above 0.3 to avoid cluttering (Fig 5.2). Average best fitness of each set of optimizations was calculated. Dataset *[indepSpeed02]* was made and then plotted showing parameter values against speed (Fig 5.3).

Results

Optimization development showed initial fluctuations of the parameter values which was damped progressively during iterations. While only a representative plot is shown (Fig 5.1), a similar tendency was evident in all optimizations. The 20 optimizations in dataset *sampling1d5* and *sampling1d5-c80* had all appeared to converge and was stopped after 30 iterations, none of them resembled any of the others by visual inspection. Furthermore, it was found that *sampling1d5-c80* using 80 cores had a mean value of 3.78, where *sampling1d5* using only 20 cores had a mean value of 4.49. Secondary results show that average simulation time when stopped after 30 iterations without improvement was: for 20 core simulation 23.25 hour with a Standard Deviation (std) of 12.39 hours and for 80 core simulation of 24.97 hours, std of 7.21 hours. It was also found that parameter variation stay within a narrow range compared to the actual bounds as specified in the auxiliary files (See Section A).

Summary

It appears that multiple repetitions of the same initial conditions result in unique parameter combinations, beyond a mere tolerance. This suggest that there exist a large number of local minima in which the optimizer gets caught. The difference between these can be explained by the stochastic properties of the CMA-ES algorithm. The difference in average Optimization development suggested that the optimal solution is sensitive to number of cores independent of simulation time. This means that a solution found by a 20-core optimization will usually result in a worse fitness than a 80-core optimization run even when overall core-time is accounted for. The reason behind this was likely caused by variable population size which in *PredictiveSim* is based on number of cores used for optimization. The *PredictiveSim* initiates by default

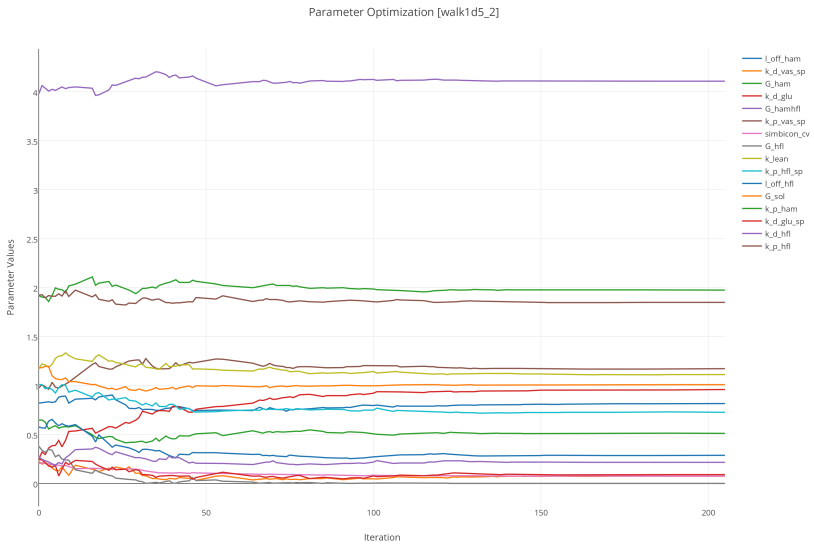

Figure 5.1: Representative plot of parameter value development through optimization. Only showing the 16 parameters with highest variance out of the total of 70 optimized parameters

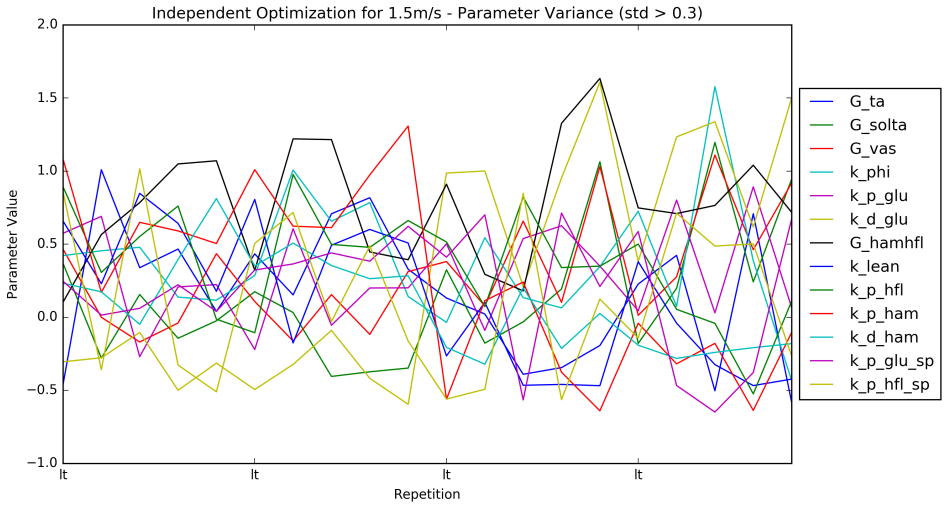

Figure 5.2: Parameter variation of multiple independent optimizations with identical initial conditions. Including 20 parameter sets, shown on the x-axis

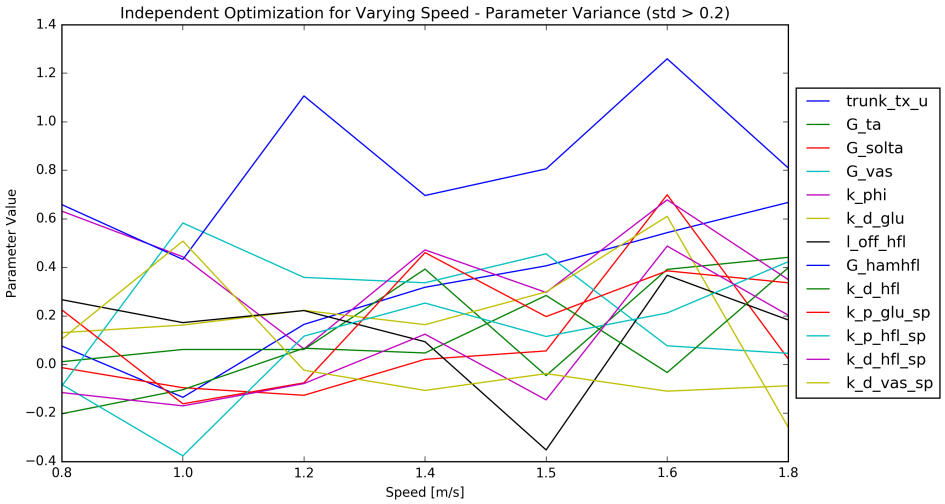

Figure 5.3: Parameter variation of multiple independent optimizations with varying speed. Only showing parameters with value $std > 0.2$

a population that is one less than the core count. This was thought to be for optimization purposes where such a method makes sure that each core only simulates once before it returns. The remaining core is allocated as "master" to manage the other "slave" cores. A high number of cores thus result in a larger population, which can perform a denser search and find more alternative options simultaneously. For this reason core numbers will be used interchangeably depending on intention to optimize use of available resources. Important results should be optimized using 80 cores where subsequent extensions on those may be done using less, usually around 20.

A *new solution* is defined as a set of parameters that shows no obvious pattern relating it to other solutions of similar fitness. Each *new solution* that solves the same problem using different parameters was then defined as Solution Strategies. This means that each Solution Strategy defines a means of locomotion that uses a varying approach to reach a similar performance.

5.2 Comparing CMA-ES and Blind Random Search

Motivation

The CMA-ES optimization algorithm came bundled with *PredictiveSim* and has been used for several other studies similar to this. However, with only few notes on why this algorithm was chosen, a method was sought for exploring its benefits as well as potential disadvantages.

Method

Used Datasets

- *cmaesBig*: Optimizations: 1, total samples: 394.163, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.
- *brsBig*: Optimizations: 1, total samples: 495.330, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: BRS.

- *brsBig-fittedRange*: Optimizations: 1, total samples: 85.642, range fitted to CMA-ES search width, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: BRS.

A comparison was made against the Blind Random Search algorithm (BRS) that was described in Section 4.2.4. Using the new feature of saving out all simulation data (See Section 4.2.1) both a CMA-ES and a BRS optimization was performed (dataset *cmaesBig* and *brsBig*). Their best was fitness was then compared. Inspired by the previous observation that CMA-ES tend to stay within a narrow range of parameter variation when looking for new solutions, the range for BRS was sought narrowed down equally. This was done by setting the bounds of BRS according to the maximum and minimum observed value for each parameter in the above CMA-ES optimization where all simulations were saved. With the new bounds another optimization was done using BRS (dataset *brsBig-fittedRange*) and the results compared to CMA-ES. Finally the bounds were sought narrowed further. This was done by filtering the new BRS results so that only simulations with fitness values below 30 was kept. The new bounds were then set based on the observed maximum and minimum values of this new data set. Due to the stochastic nature of both algorithms both were compared by histograms of fitness, showing the full range of fitness along with a zoomed in view between 0 and 30.

Results

Creating the *cmaesBig* dataset resulted in a total of 394.163 saved simulations with best fitness of 3.8. *brsBig* resulted in a total of 495.330 simulations with best fitness of 92.5. Using the overall maximum and minimum of each parameter from *cmaesBig*, a new set of search bounds were made with a narrower range, as seen on Fig 5.4. Using these bounds for another BRS optimization resulted in *brsBig-fittedRange* with 85.642 saved samples and a best fitness of 6.9. The further narrowing of bounds using fitness less than 30 provided a difference of less than 0.1% and for the most parameters much less. No further optimizations were therefor made. From the histograms of BRS (Fig. 5.6) and CMA-ES (Fig. 5.5) it is seen that CMA-ES has a much higher density of samples in the lower end of the scale, where BRS is performing many more simulations that result in high fitness values.

Summary

The performance comparison between CMA-ES and BRS suggest that CMA-ES offers a significant improvement over BRS when looking for the fastest way to good fitness. If the bounds of BRS are narrowed down to match the search space of CMA-ES, then performance increases significantly with BRS though still not matching CMA-ES. However, the fact that BRS searches a much wider range by default may also prove beneficial. The less than 0.1% difference of second iteration for narrowing bounds suggests that a fitness values of 6.9 can be achieved even at the very borders of all parameter bounds. For this reason further investigations should be made in order to correctly narrow down the search space. It is still possible that good fitness can be achieved outside those bounds using a radically different strategy. However, it is assumed that the current resulting strategies resemble human walking enough, so that radically different strategies can be disregarded.

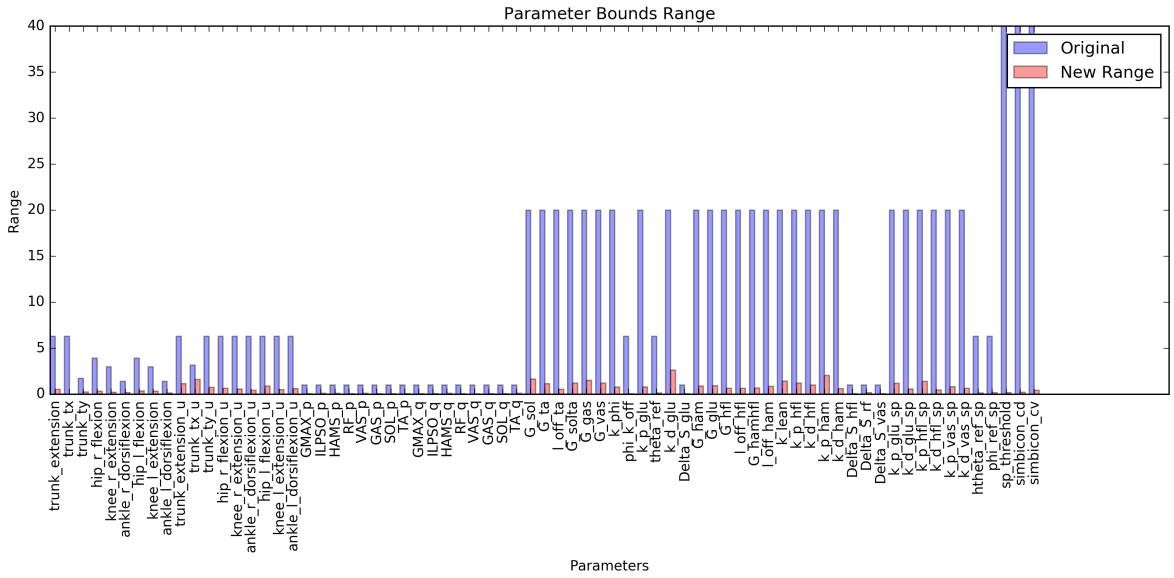

Figure 5.4: Specified parameter bounds compared to actual search width of the CMA-ES algorithm. Based on 350.000 simulations

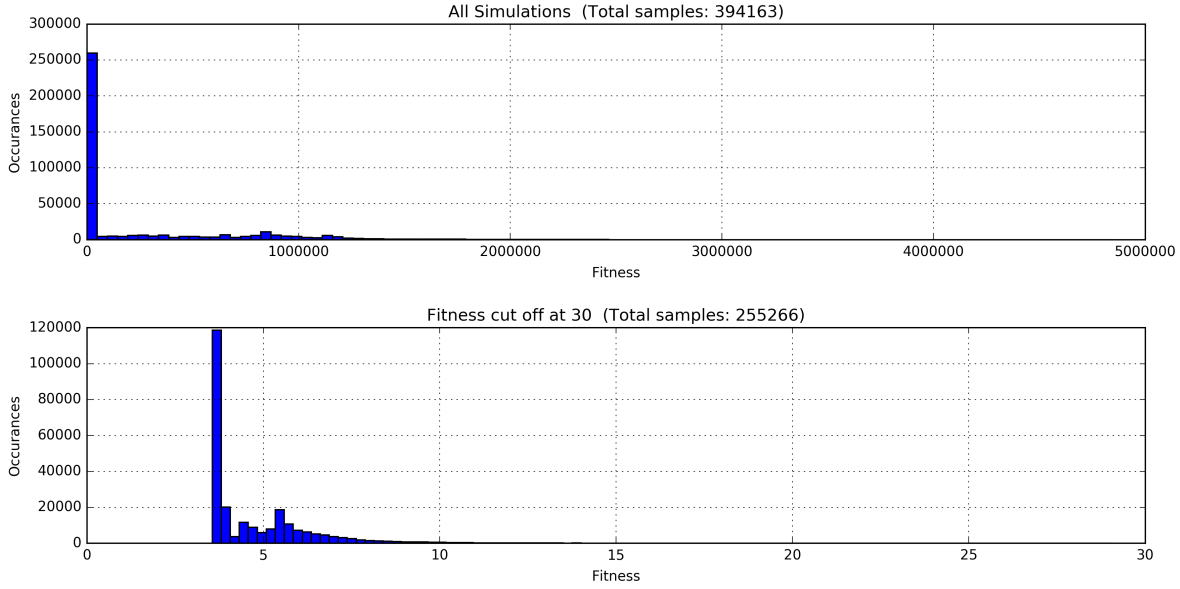

Figure 5.5: Histogram of Covariance Matric Adaptation. All simulations included

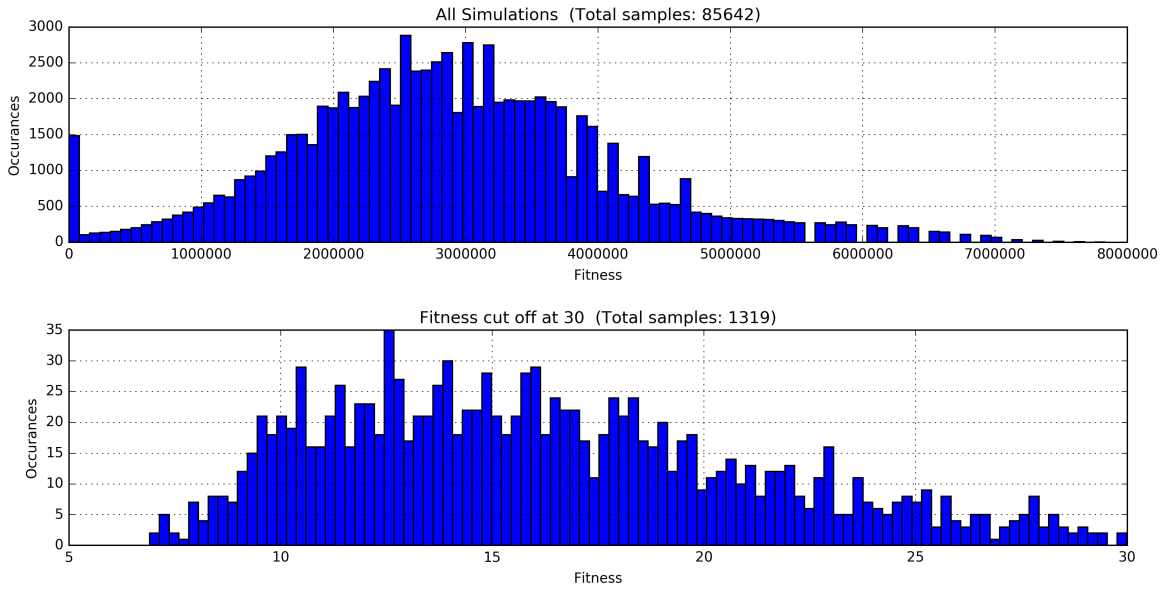

Figure 5.6: Histogram of Blind Random Search with fitted search bounds

5.3 Differentiating Solution Strategies

Motivation

Based on the discovery that all solutions appeared to be unique (See Section 5.1), a strategy was sought to differentiate the different solutions and investigate how and why they vary.

Method

Used Datasets

- *sampling1d5*: Optimizations: 10, cores: 20, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.
- *sampling1d5-c80*: Optimizations: 10, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.

From the dataset *[sampling1d5-c80]* the 10 optimum solutions were simulated independently for 8 seconds to generate motion data of the musculoskeletal model. It turned out that 2 solutions could not simulate properly. Therefore only 8 were actually used. In order to only use data after the simulation had stabilized a script was made to find the last whole step of the right foot and mark the start and end. This range was then plotted from right heel strike to right heel strike. Due to varying sample length all samples were truncated to match the shortest meaning that not all samples were plotted all the way till the next heel strike would have appeared. These data were then plotted for all 8 strategies, showing the impact of the different solution strategies. Ground Reaction Force (Fig 5.9), Kinematics (Fig 5.10), Torque and Muscle Activation (Fig 5.11). The differences of each strategy were also visualized by performing a PCA on all simulation data coming from those 8 optimizations (See Fig. 5.7).

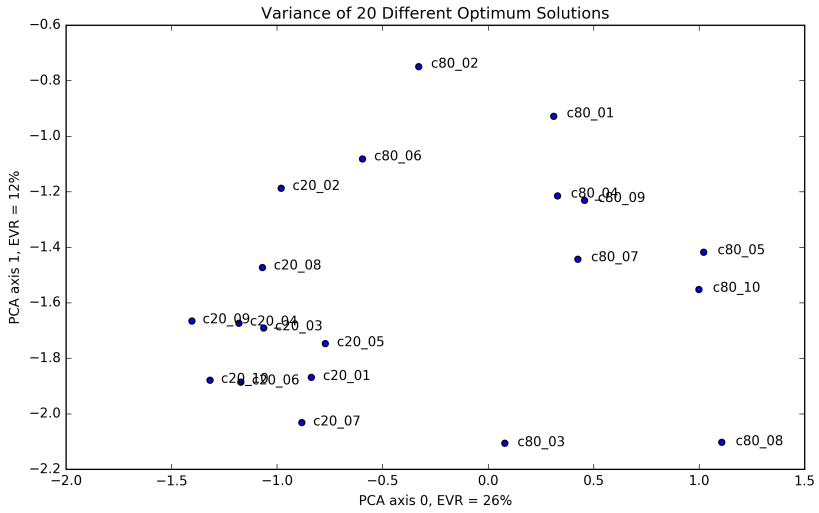

Figure 5.7: 2D PCA plot of the same samples showing that the are all different

To investigate the reason behind the different strategies Principal Component Analysis (PCA) was used to plot the development of optimizations (See Appendix B.1.2 for details on PCA). This was done by performing a PCA on all control files from dataset [the sampling1d5 and sampling1d5-c80]. Each sample was then transformed by the PCA and plotted on in a 3D plot with the 2 principal axis' and their fitness value on the last (Fig 5.8).

Results

The Kinematics, Torque and Muscle Activation varies in both shape and time across the different solution strategies. It was furthermore found that simulation data did not have consistent internal time steps. This made it difficult to quantify their differences and plot them correctly. No ideal solution was concluded for normalizing simulation data correctly. For that reason the data was plotted as-is but truncated to match length.

During simulation of each sample control file it was found that fitness values were not consistent between logged fitness from optimization and performing an independent simulation.

A difference of about 0.02 in fitness was found between results from the optimization and independent simulation. Beyond that control files that had been labelled with a good fitness turned out to stumble and fall when simulated independently thus never achieving a stable gait. This was the case for the two files that could not simulate. The problem was furthermore verified by discussions with other users of the software.

Summary

Fig. 5.8 suggest that each strategy gets formed by the first couple of iterations scattered in the back of the plot. The best of these initial stochastic samples then points a direction for the CMA-ES algorithm to travel. The optimizer then narrows down its search width progressively. This eventually traps it in one path that it will continually improve upon. These data show the varying gait patterns resulting from the optimizer finding different local minima for each

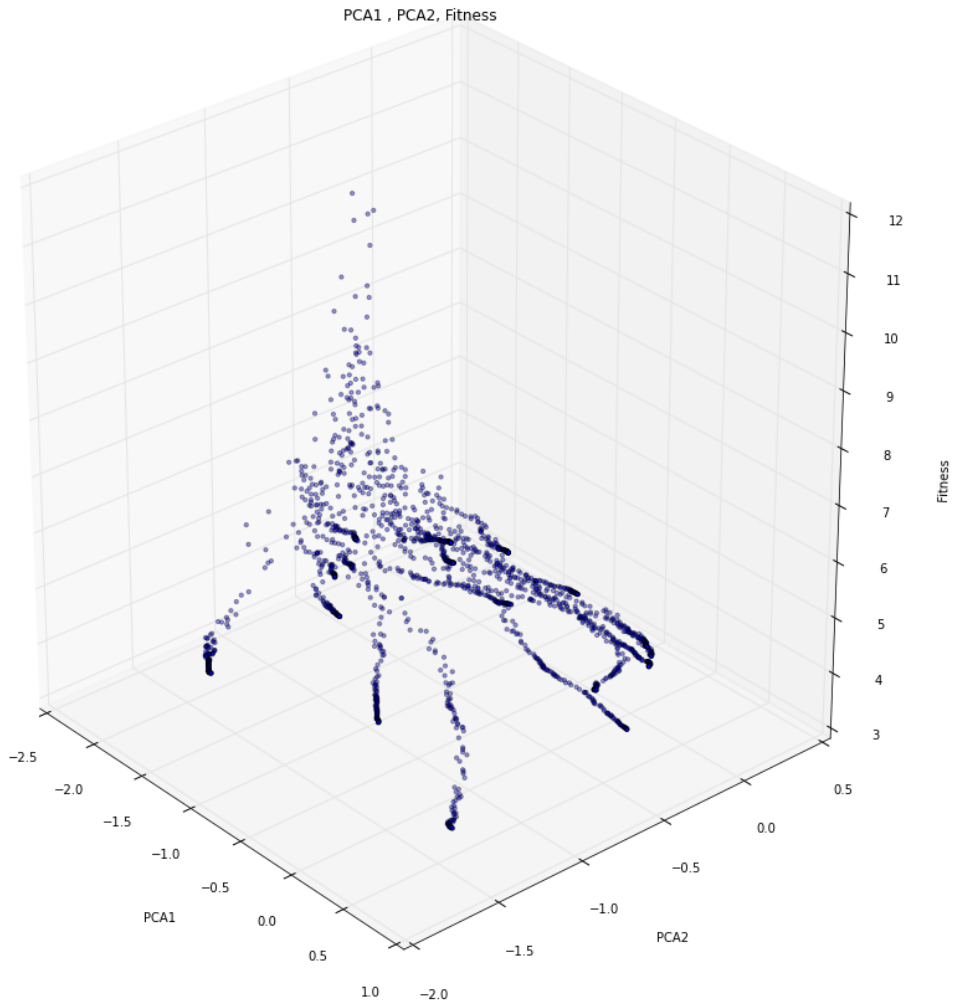

Figure 5.8: PCA of control parameters development for 10 different solutions with identical conditions. Showing only datapoints with consecutive improvements

optimization. The plotted motion data only really allow for an overall impression of the data but invalidates actual comparison by small variations due to the difference in time steps.

Rounding off floating point numbers when saving control files was found as a possible cause for the inconsistent fitness between optimization and individual simulation. This could give slightly different values control files were loaded compared to the internal values used for simulation during optimization. Current result files are printed using 21 decimal places, but further investigations are required to determine the internal floating point representation in OpenSim and whether that could be the cause. The fitness value is an abstract measure and data resulting from this thesis is not meant for clinical use or as conclusive evidence. For that reason and because others had failed to locate the error, this issue was not thoroughly investigated and deemed outside the scope of the present study. However, this introduces a random error that could result in significant differences which are difficult to estimate.

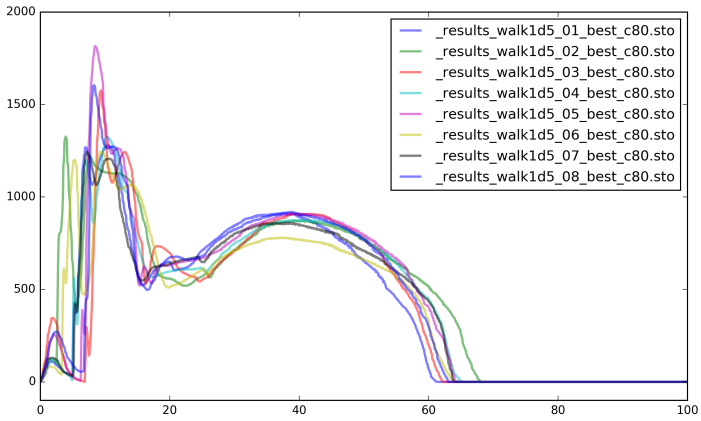

Figure 5.9: Step Cycle Ground Reaction Forces, 9 optimization repetitions for 1.5m/s, optimized using 80 cores

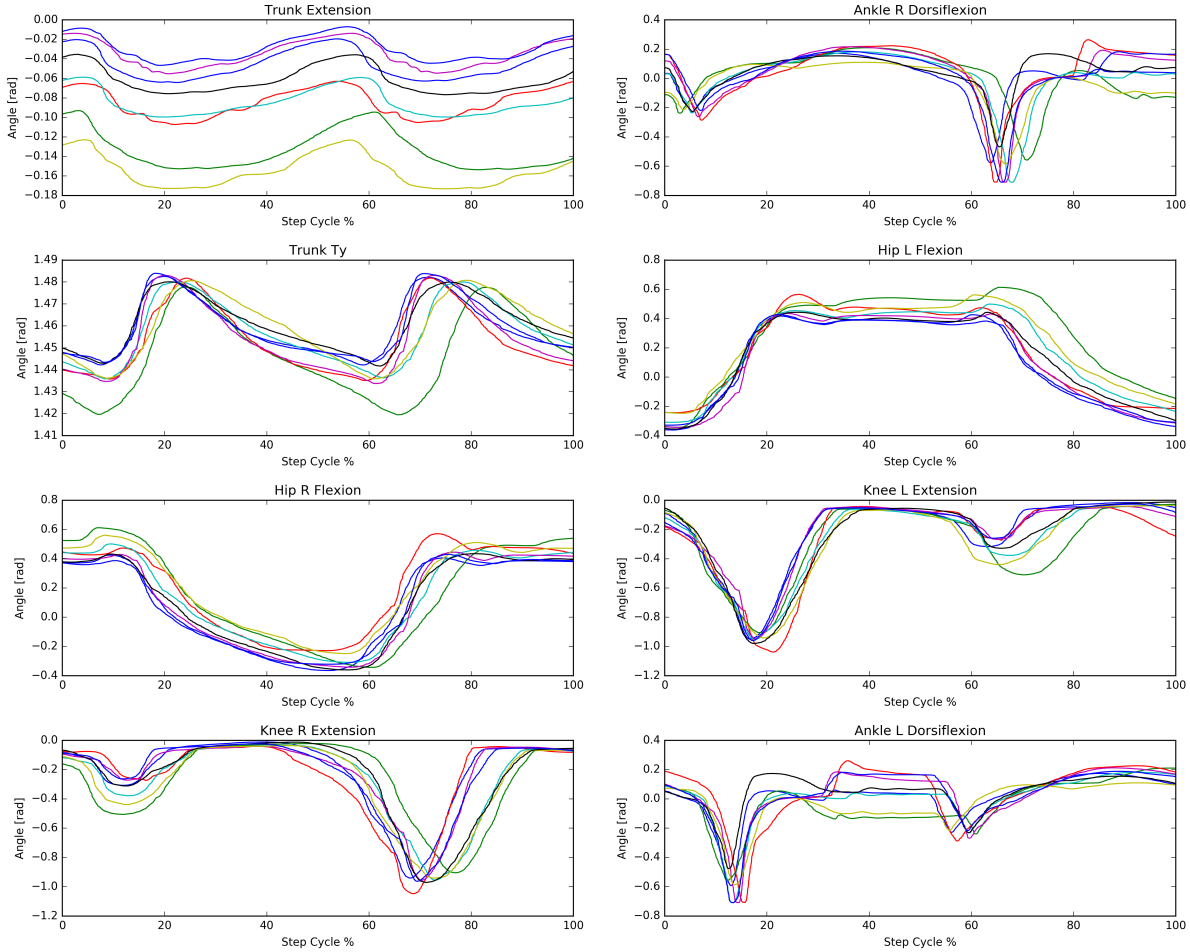

Figure 5.10: Step Cycle Joint Angles, 9 optimization repetitions for 1.5m/s, optimized using 80 cores

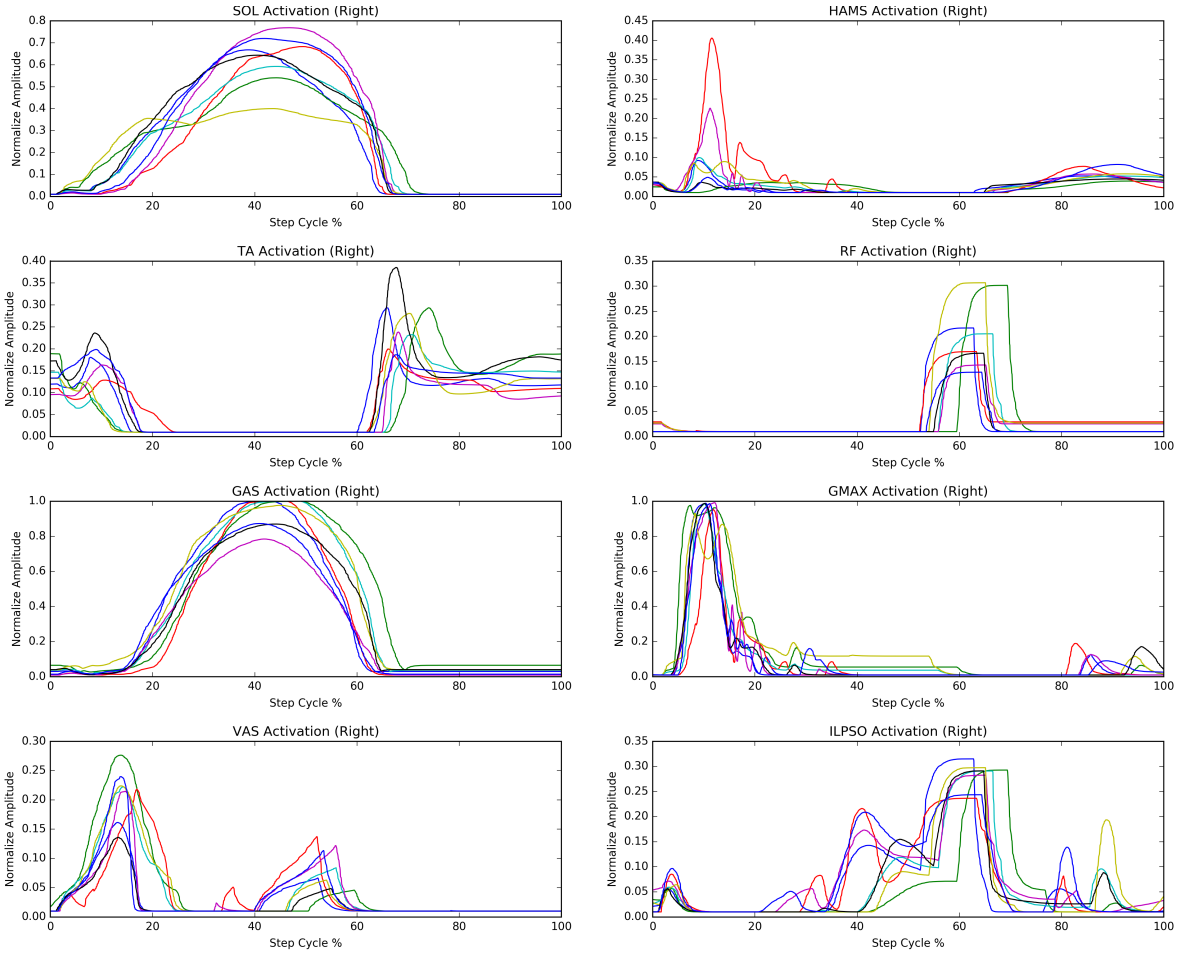

Figure 5.11: Step Cycle Muscle Activation, difference in activation of 9 optimization repetitions for 1.5m/s, optimized using 80 cores

5.4 Exploring Stability, Speed and Control

Motivation

It was found that the reflex-controller is able to adapt to varying targets by changing its parameters. This includes handling a range of different perturbations and speeds by optimizing for each argument and finding a new set of parameters. Using this principle several features and questions appear, including how simulations behave when changing parameters and how transitioning between multiple parameter sets are handled. The following experiment explore these questions. The following method relates to the optimum control file found by a certain set of input arguments by stating their argument, ie. using 1.5m/s means using a control file optimized for a target velocity of 1.5m/s and 0.0 perturbation amplitude. Similarly using 0.6 perturbation amplitude at 1.5m/s refers to the control file found by optimizing for those parameters.

Method

Used Datasets

- *consecSpeed*: Optimizations: 7, consecutive optimization, perturbation: 0, cores: 20, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.

- *consecPerturb*: Optimizations: 7, consecutive optimization, perturbation: from 0.0 to 0.6 in steps of 0.1, cores: 20, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.

Online changing of speed was explored during several simulations of different control files. The following three simulations all used optimum solutions from dataset *consecSpeed*. First a simulation was started at 0.8m/s and was set to switch to 1.8m/s after 4 seconds. This was then simulated for 10 seconds in total. Similarly another test was made which went from 1.2m/s to 1.4m/s using same timing. Then a third test was made using several steps from 1.0m/s to 1.8m/s in steps of 0.2m/s. The first switch was similarly applied after 4 seconds and the speed was then further increased every 2 seconds, simulating for 20 seconds total.

The impact of perturbations in optimization was then explored. This was done by performing 2 simulations both using dataset *consecPerturb*. First a simulation was made starting at 1.5m/s with a perturbation amplitude of 0.0 and switching to 1.5m/s and an amplitude of 0.6 after 4 seconds, simulating for 10 seconds total. Then a simulation was started at 1.0m/s with no perturbation and changed to 1.5m/s with a perturbation amplitude of 0.6 after 4 seconds, simulating for 10 seconds.

Results

The initial test of going from one extreme speed (0.8m/s) to another (1.8m/s) stumbled and fell almost instantly after switching parameters. The next test where the transition was much smaller resulted in a stable gait defined by the new parameters, thus causing an acceleration of 0.2m/s. By applying several parameter updates consecutively, the simulation stumbled and fell, despite the small individual steps. Changing from no perturbation to a perturbation amplitude of 0.6 resulted in a stable gait, which by visual inspection mainly concerned a variation of the trunk angle, making it more vertical. Going from one speed to another that was optimized with perturbation allowed for a larger speed transition and resulted in a stable gait, thus inducing a speed increase of 0.5m/s.

Summary

When performing supraspinal control and going from one set of parameters to another, it was assumed that switching between different solution strategies could also impact the stability. Due to this and the varying impact of switching based on timing, as well as the varying timing of each simulation, it was found impossible to properly quantify this measure within the time frame of the present study. Further studies will therefore be needed in order to properly specify the impact of using parameters with high stability for changing speed. It was furthermore noted that the time at which the parameters change could have an impact on the resulting stability.

5.5 Predicting Fitness and Correlating Parameters

Motivation

Predictive Simulation systems like *PredictiveSim* consist of a complicated interrelated circuitry that relies on a number of abstract parameters. These parameters are often implemented in a purely functional way which is difficult to relate to. Furthermore the parameters are often difficult to find, requiring high performance computers and many hours of optimization for each individual set of conditions. For that reason any improvement in performance for finding good fitness values are interesting, thus exploring potential high-level patterns could prove both valuable for optimizing time, but also in relation to Supraspinal Control, where parameters could be made to change based on such patterns. Either independent of optimization or using more narrow search criteria.

Method

Used Datasets

- *sampling1d5*: Optimizations: 10, cores: 20, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.
- *sampling1d5-c80*: Optimizations: 10, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: CMA-ES.
- *brsBig-fittedRange*: Optimizations: 1, total samples: 85.642, range fitted to CMA-ES search width, cores: 80, step size: 0.005, target velocity: 1.5m/s, time: 10s, Optimizer: BRS.

Approaching this was here done by Machine Learning to investigate whether such algorithms can see a pattern that is difficult to find otherwise. Using the Python SKLearn library (Pedregosa and Varoquaux, 2011) three linear regression models were done, training on three datasets: *Sampling1d5*, *sampling1d5-c80*] and *brsBig-fittedRange*.

(1) Dataset *sampling1d5* and *sampling1d5-c80* combined were used to train the algorithm. This was done repeatedly for 100 iterations, training on 90% of the data and using the other 10% randomly chosen samples to validate, yielding an average R^2-value. (2) The same was then done again on the same dataset but this time using one whole optimization (solution strategy) as validation data. By training on 19 optimizations and validating against the last, an average R^2-value was found for predicting across solution strategies. (3) Dataset *brsBig-fittedRange* was analysed similar to the first using randomly chosen validation samples. This dataset was furthermore truncated to only contain samples with fitness below 30. The value yielding the best correlation was found by a simple scripted loop, which resulted in the best fitness cutoff as well as a third R^2-value.

Results

(1) gave and average R^2-value of 0.93 validating against 10% random samples.. (2) gave an R^2-value of 0.36 for predicting across strategies. (3) was found to provide the highest R^2-value when truncated at $fitness < 30$. This reduced the set to 1319 samples resulted in an R^2-value of 0.26.

Summary

First R-value showed predictions inside a large amount of samples. This expresses that linear regression can approximate the interrelated position of the samples, which is not a big deal. The R^2-value of 0.36 is too low for any conclusions. However, refined methods for pre-treating the data before training, or exploring non-linear models instead may be able to find a better correlation. The value of fitness cutoff at 30 is in accordance with the pattern seen on fig 5.6 where the samples seem to appear in two groups, separated at a fitness value of 30. While these numbers do not show an impressive correlation the results suggest that there may be benefits of applying machine learning algorithms to both analysing and predicting the solution space. SVM or other techniques should therefore be tested further to see if better correlation can be achieved. Either way this shows that swiping through the solution space using algorithms such as BRS while saving all simulation data may provide a foundation for estimating where to look for good fitness in a less biased way. This could potentially be used to supplement optimization algorithms to make more qualified guesses.

5.6 Parameter Variation Tendencies

Motivation

Based on the fact that Predictive Simulation systems presented here are founded biology, there may be benefits in analysing the parameters themselves. The parameters are able to vary to accommodate a range of different tasks. It is then hypothesized that gaining insight into this structure might potentially reflect in clinical observations by being approached from a different angle.

For exploring parameter variations it was hypothesized that the difference caused by varying input arguments are larger than the potential difference between solution strategies. In this way a range of optimizations with varying arguments could show statistical tendencies that correlate better with variation of the argument in question rather than solution strategies.

Principal Component Analysis is a method that finds the combination of parameters that linearly express the biggest variation in a dataset. Therefore, if the above hypothesis is correct, then the first principal axis of a PCA could express this relation between parameters and the corresponding input argument. The hypothesis can then be validate if the samples with varying arguments appear ordered on the first PCA axis according to their argument value. If the relationship is not linear, then multiple PCA axis may express the argument variation combined.

Method

Used Datasets

- *consecSpeed-smallStep*: Optimizations: 11, repetitions: 3, consecutive optimization, perturbation: 0, cores: 20, step size: 0.001, target velocity: 0.8 to 1.8m/s with increment of 0.1m/s, Optimizer: CMA-ES.

- *consecPerturb-smallStep*: Optimizations: 6, consecutive optimization, perturbation: from 0.0 to 0.6 in steps of 0.1, cores: 20, step size: 0.001, target velocity: 1.5m/s, Optimizer: CMA-ES.

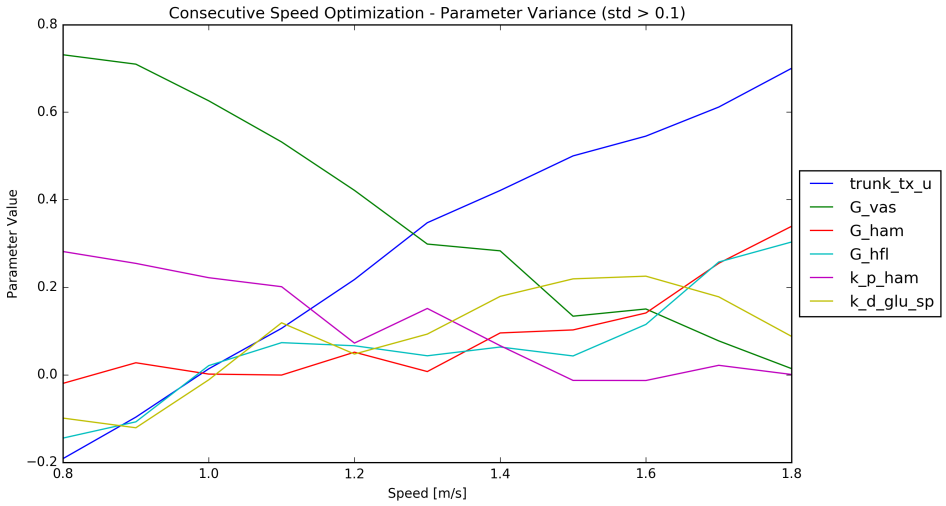

Figure 5.12: Parameter variation of multiple consecutive optimizations with varying speed. Showing only parameters with $std > 0.1$

- *indepSpeed02*: Optimizations: 7, cores: 20, step size: 0.005, target velocity: from 0.8m/s to 1.8m/s with increments of 0.2m/s, Optimizer: CMA-ES.

Consecutive optimizations were used, applying a small step-size and small increments for each new optimization. This was done to minimize diversity by solution strategies for easier comparison and analysis.

The resulting optimized parameter sets were compared while looking for tendencies in parameter development related to the varying speed or perturbation arguments. PCA was used to quantify variation according to which parameters change with varying arguments.

First, repetition one of *consecSpeed-smallStep* was plotted with each optimum solution on the x-axis in consecutive order of speed, showing their parameter values on the y-axis, truncated by $std > 0.1$ to avoid clutter (Fig. 5.12).

PCA was then applied to *consecPerturb-smallStep* and plotted on the two first principal axis' (Fig. 5.13)

The same was done for repetition one of *consecSpeed-smallStep* as well as *indepSpeed02* plotting the two first principal axis accordingly: Fig. 5.14 and Fig. 5.15

Finally all 3 repetitions in *consecSpeed-smallStep* were analysed. Their data was combined and plotted by PCA (Fig. 5.16).

Results

The parameter variation of speed with small step-size shows some tendencies that could correlate with varying speed, yet still lacking clean variations (Fig. 5.12).

Perturbations : For a single repetition of consecutive optimization from 0.0 to 0.6 in amplitude the first two axis' of the PCA were found to generally order each sample in accordance with the increasing amplitude argument. Only exception was the two samples with amplitude of 0.0 and 0.1 there were almost coinciding. These axis' were also found to explain 56% and

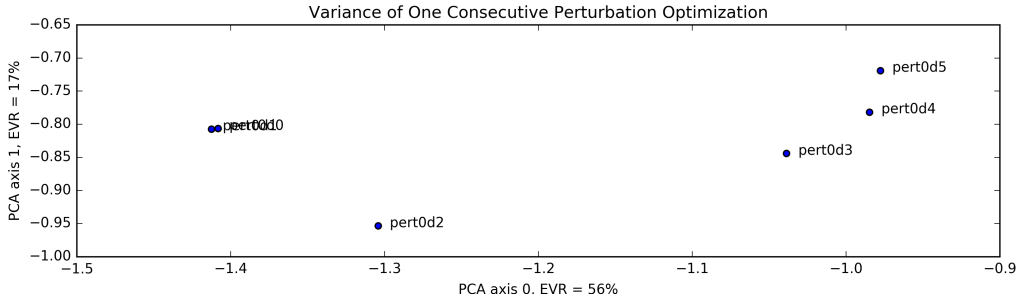

Figure 5.13: PCA of dataset [perturb from 0-0.6]. The combination of the two best axis of the PCA appears to order the different samples according to their varying perturbation amplitude. The two samples that are on top of each other are for amplitude 0.0 and 0.1

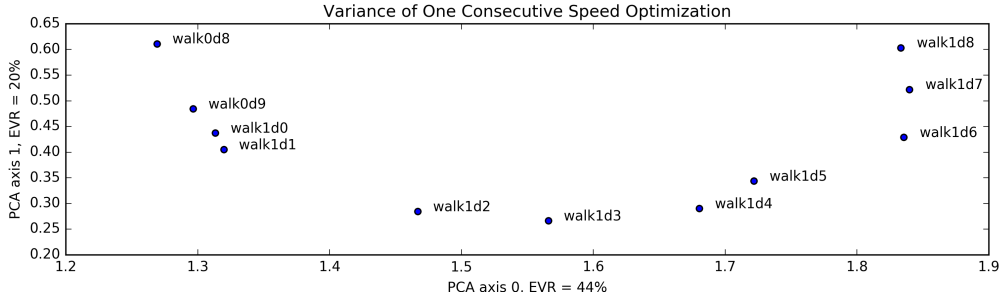

Figure 5.14: PCA of first repetition of *consecSpeed-smallStep*. The data is plottet on the first two axis' of the PCA. The two PCA axis' appear to separate the individual samples in the same order as their target speed, which combined explains 64% of the overall parameter variance.

17% respectively of overall variance. Figure 5.13 shows the samples plotted on the first and second axis of the PCA.

Speed : Fig. 5.14 shows a PCA plot of *consecSpeed-smallStep* where samples are sorted according to target speed. The PCA plot of *indepSpeed02* shows a similar pattern is seen, despite each optimization is performed independently (Fig 5.15).

Summary

The essence of using PCA in this way is that the algorithm finds a list of weighted parameter sets that by linear combination can explain the variation in the samples. The first set explains the most with all the following explaining less and less of the variance. In this way, looking at the primary set/axis, you get a sense of how the parameters vary together in order to go from one sample to another. Ie. Fig 5.17 shows the parameter combination that forms the first PCA axis of Fig. 5.13. By adding this axis to the sample parameters in different scale, 56% of their variance can be achieved. Between sample "pert0d2" and "pert0d4" appears to be about 0.3 in difference on the x-axis. This means that 56% of the difference between those two sample can be accounted by for adding $0.3 \cdot PCA1$ to "pert0d2". By using the second axis as well,

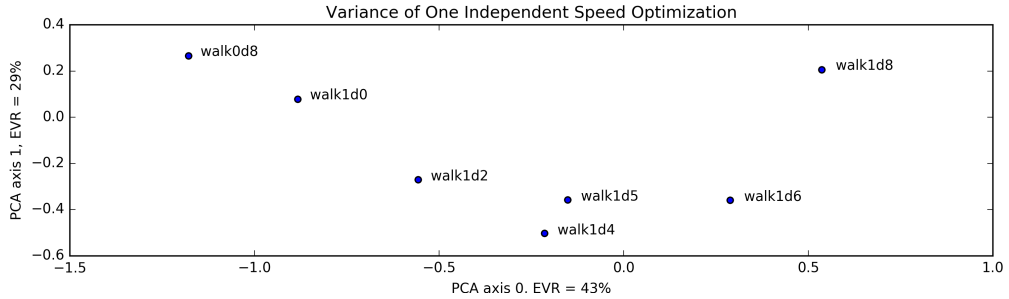

Figure 5.15: PCA of dataset *indepSpeed02*. Labels describing target velocity, 1d2=1.2m/s. The two PCA axis' order the samples according to the varying target velocity, despite all samples being individual optimizations.

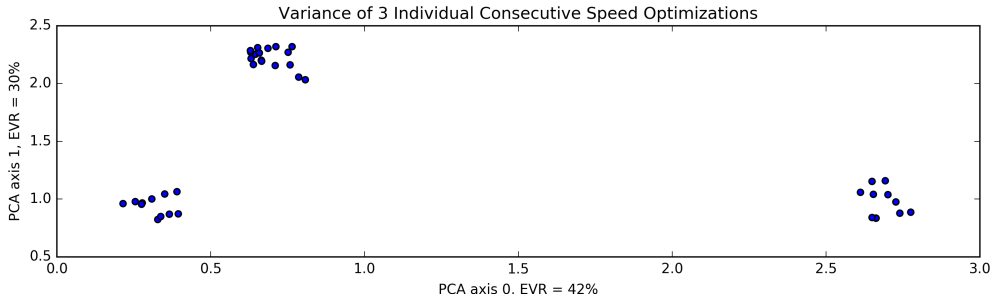

Figure 5.16: PCA of all repetitions from *consecSpeed-smallStep*. Each cluster of samples correspond to an individual optimization run, where each individual point is a different target speed

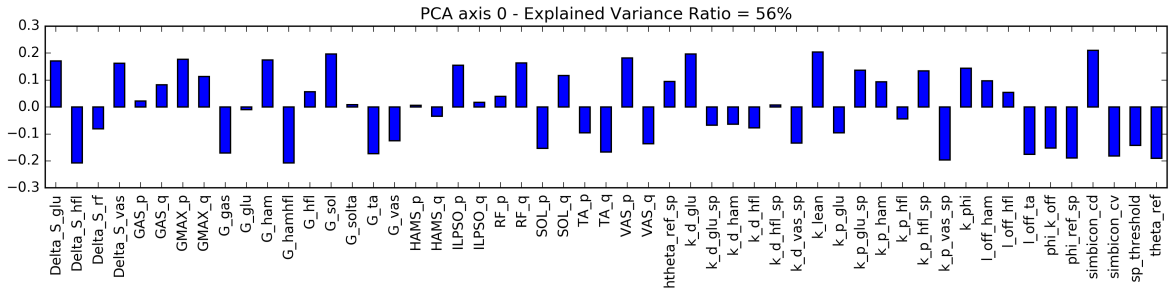

Figure 5.17: Explained Variance Ratio of the primary axis, dataset [perturb from 0-0.6]. Showing the linear combination of each optimized parameter that form the primary axis and explains 56% of the overall variance for dataset

another 17% can be accounted for. However, this approach only shows the combination that by a linear approach can explain the most variance and does not describe everything. The linear structure of this also means that the explained variance works best for parameters that vary linearly. If they for example vary exponentially instead then several additional axis are required to properly explain a variance by a linear combination. This will make the result less clear. As seen in fig 5.14, some parameters does indeed appear to vary along both the 1st and 2nd principal axis which suggests that the parameters have a non-linear relationship to varying speed.

The high amount of variance explained by a single axis as well as the automatic ordering according to amplitude or speed could suggest that the algorithm found a good indicator for the variation of parameters that obtain different stability and speed. Additionally the PCA plots of single optimization runs show an interesting tendency on the 2nd PCA axis, especially in those made by consecutive optimizations (Fig. 5.13 and 5.14). The independent optimization run shows similar tendencies yet a little less structured. However, this run was not intended to maintain any particular solution strategy meaning that all samples are using a new and unique strategy to achieve stable gait at each different speed. The implications of this are that the varying speed appear to result in a bigger variance than the individual solution strategies.

When PCA is applied to dataset *consecSpeed-smallStep* containing three different strategies, however, then the variance between each strategy appear to outweigh the variance governed by change in input arguments (Fig. 5.16). Alternatively this may suggest that different strategies vary their parameters differently for achieving the same target. In this way the variation caused by speed change from one strategy may be different for another strategy.

The resulting parameter weighting in the EVR plot (Fig. 5.17) could then be a potential method for extrapolating parameters to reach new parameter combinations capable of handling situations that have not been optimized. In this way it could be interesting to use these PCA axis' for generating new parameters that can be simulated. They may also help in predicting good fitness for a certain combination of arguments that have yet to be explored by optimization.

This result could also prove beneficial for parameter interpolation in supraspinal control (See Section 5.4).

5.7 Running

It was attempted to get PredictiveSim to realize a running gait by following notes in both Geyer and Herr (2010); Wang et al. (2012) by editing initial parameters but without success. The controller did not seem capable of switching strategy and only managed a more bouncy gait which realized a much slower speed than targeted. A conversation with Jack M. Wang suggested that this could be due to the 3-stage setup, where Geyer and Herr (2010); Wang et al. (2012) both used a 4-stage setup. However, the 4-stage method was found to be less efficient in Dorn et al. (2015) which caused the change.

Chapter 6

Further Thoughts

This Chapter adds a few last notes to the material that has been presented up till this point. These concern Structure of parameter value variation, the multiple solutions that evaluate with equal fitness, connectomics of controllers, current bottleneck as well some benefits and drawbacks of adding complexity.

Structure in Parameters: Looking for structure and patterns in the parameters can from here initially be approached by simple clustering algorithms. However, it is also hypothesised that such patterns could be approached by Deep Learning algorithms or other methods, in the hopes that high-level structures may appear. This may also eventually form the basis of the supraspinal control, that could be driven by such overlying patterns to constantly modulate the reflex loops (Lobo and Levin, 2015).

Multiple Equal Solutions: The data suggests that there are a very high number of solutions that are deemed equally good by the fitness function while possessing slightly different strategies. This can suggest two different points: One is that the Objective Function should be improved to make a more concave solution space to allow the optimizer to find the global optimum. Alternatively, the goal may not be a single optimal solution after all, if either of the solutions are equally good. This could be viewed as a potential reason behind the unique body language of humans. It is possible that such shape of the fitness function is not completely implausible and that each unique gait stems from the human nervous system stranding on a local minimum, thus adopting a particular gait strategy from which future learned behaviours spring. Furthermore this may explain why running-trainers, with an analytical approach to gait patterns, can train a human by proper instructions, to adopt a more efficient pattern that the body did not find by itself. Research therefore needs to be done to explore this further. This could potentially be done by comparing the different strategies and looking for potential ways to evaluate their differences and maybe determine if and how some are better than others. Such an observation could be added back in to the Objective Function which would then be able to better determine good strategies from bad.

Connectomics of the Controllers: By introducing the idea of multiple pathways that are activated or inactivated at different times, a diverse locomotor system could in theory be

established by an high number of reflex-loops with corresponding high amount of state-specific parameters. As shown by Jeff Lichtman, however, neuromuscular connectomics show that in mice there is constant competition between axon development where less used pathways decay while active pathways reinforce themselves (Tapia et al., 2012; Coggan et al., 2004). This suggests that a mature body can be expected to posses only the most used neural pathways. It could then be hypothesized that this follows an optimization of both energy and real-estate where diversity of movement is achieved by the least amount of connections possible. How this diversity is defined could prove a valuable point when approaching design of biologically inspired Predictive Controllers. This method may even be directly applied as an optimization process for designing controllers. See Future Works for more (Chapter 7).

Bottlenecks: Some of the bottlenecks for progressing to a more life-like synthetic motion appears to be located in both the design of the controller and formulation of the objective function. Now that we have reached stable gait patterns and thus verified that this is possible using relatively simple systems, it is therefore important to approach the designs of both controller and objective function in a way that minimize bias.

Added Complexity: State-of-the-art Predictive Controllers provide a good basis for further research where the controllers iteratively can be moved towards a more physiologically plausible system while seeking to get progressively closer to clinical data. Several of such extensions are already under way including (Song and Geyer, 2015), (Dzeladini et al., 2014) and (Geijtenbeek et al., 2013). However, while their effort in many cases has added further functionality which is believed to improve biological feasibility, the resulting motion does not always show improved resemblance with human motion.

Chapter 7

Conclusion

Predictive Simulation in biomechanics is currently able to successful achieve gait by optimizing for high-level objectives, both in 2D and 3D. The resemblance with humans varies across the different controllers but overall they are approaching one standard deviation of clinical data. Despite this, some features persist in being elusive and functional diversity is in general limited, so that elaborate and alternative formulations of high-level objectives are being explored along with the connectomics of the controller.

During this thesis an overview of Predictive Simulation was given, establishing a theoretical foundation for approaching the topic. Overall functionality of Predictive Simulation is presented followed by a description of its sub-systems. These include: Predictive Controller, Mechanical System, Simulator, Objective Function and Optimizer. By the combination of these sub-system it was possible to perform an optimization seeking to minimize the output of the Objective Function. The results of this was a motion pattern of a human-like virtual model that appears to walk similar to humans. Following this introduction a broad overview was given of the human Nervous System, which forms the foundation of biologically inspired Predictive Controllers for human gait.

Subsequently the software *PredictiveSim* was explained. This is a free software package for performing Predictive Simulation, made by Tim Dorn and Jack Wang at Stanford University (Dorn et al., 2015). Additional functional features were implemented including: Adding the ability to vary controller parameter during simulation by supraspinal control, adding perturbations so that optimizations can be quantified by stability and finally adding Blind Random Search as an alternative optimization algorithm.

The extended software was used for making experiments on human gait, analysing optimized solutions and exploring patterns in the 70 optimized parameters yielding different features when simulated. The experiments led to different conclusions that are explained as follow.

Initially the ability of *PredictiveSim* to synthesize a gait pattern without recorded data was verified and that it by visual inspection resembles human walking. This is in accordance to what was found by Dorn et al. (2015). *PredictiveSim* uses a simplified human model with 6 joints, 9 DOF and 16 hill-type muscle and a controller. The controller is defined by a set of reflex loops relating foot contact and muscle states to muscle activation. The controller is parameterized by 77 values of which 70 are optimized by an iterative stochastic optimizer (CMA-ES). The resulting gait pattern is then a consequence of a minimization problem, which seeks to decrease metabolic cost per distance travelled while achieving a certain speed.

Getting the system to synthesize a running gait was briefly attempted but without success, possibly due to the definition of controller states, which here differs from the other similar controllers that were able to run.

A discrepancy was found between the resulting fitness values of parameters when optimized and the fitness value yielded by subsequently simulating the same parameters by loading the control file. The discrepancy was found in some cases to be approximately 0.02, yielding the higher (worse) value when the control file was loaded and simulated. In several other cases a simulation based on a control file that contained a good fitness from the optimization process, appeared to stumble and fall. A possible reason for this was an insufficient value-precision when saved to control file and subsequently loaded for simulation.

Supraspinal control was explored by a series of experiments involving changing controller parameters during a simulation. These showed that speed changes of 0.2m/s can be achieved by swapping between parameters of 1.2 and 1.4m/s. A larger speed change resulted in the simulation stumbling and falling. If instead the parameter set had been optimized for perturbations, a speed change of up to 0.6m/s was achieved before stumbling, thus suggesting that optimization with perturbations cause an increased stability margin.

Analysis of the CMA-ES optimization algorithm showed that each repeating optimization arrived to a unique solution, despite identical initial circumstances. This was further explored by investigating the path of the parameter variation through optimizations. In this case the initial stochastic sample was found to have a large impact on the subsequent path. This would then suggest a direction which initiated the optimization algorithm on this path, disregarding alternative paths that could lead to better fitness. The optimizer thus has a tendency to get caught in local minima and a more diverse strategy should be considered.

The resulting parameters from a number of optimizations were analysed by looking for structured variance relating to changes in speed or resistance to perturbations in order to explore predicting combinations that could result in so far undiscovered combinations. The analysis was done through linear regression Machine Learning and Principal Component Analysis. By training a linear regression Machine Learning algorithm on a 1319 parameter sets with corresponding fitness, the algorithm predicted a fitness outcome with an R^2-value of 0.364. The use of Principal Component Analysis allowed to find different correlations. The first and second principal axis by linear combination could explain between 64% and 73% of overall parameter variance, correlating with speed and perturbation increments.

This suggest that the computation-intense optimization algorithm may benefit from using such strategies to predict initial conditions for subsequent optimizations. Furthermore, the correlations suggest that there could exist an overlying pattern that governs these parameter variations for different tasks. Learning more about these patterns may provide useful insight, both for improving performance of optimization but also as a means to explore supra spinal control and what kind of logic may be supporting adaptive reflexes in locomotion.

Finally it was found that interdisciplinary studies are essential for approaching topics such as Predictive Simulation. Many fields that in engineering are defined separately appear to fuse on a higher level when used to approach difficult questions.

7.1 Future Works

Based on the present study a multitude of future works are required to proceed further. However, one particular stands out which is lowering bias of the controller. A hypothesis is presented

here to approach overcoming the bottle-neck that is inherent in today's designed controllers:

The method relies on 2 parts: (1) Establish a well-defined set of building block that may be connected to provide the functionality of current controllers (See Section 3.6 for an initial attempt). These blocks are then used to relate all sensory input to all muscles in various ways. (2) A simple musculoskeletal model, similar to that of the present study, is matched to motioncapture data using dynamic optimization of muscle activation. The system of (1) is then optimized for matching the muscle activation found by (2) using either or all of its sensor to muscle relations, and applying a similar strategy of neural decay as presented by Jeff Lichtman (Tapia et al., 2012). This method is slowly decaying less likely connections while emphasizing those that prove better correlation. After allowing most connections to completely decay, the product would then be a controller containing only the relations that emphasize correlation with the tracked motion, thus potentially providing new functional connections that are not easily designed manually.

Bibliography

M. Ackermann and A. J. van den Bogert. Optimality principles for model-based prediction of human gait. *Journal of Biomechanics*, 43(6):1055–1060, 2010. ISSN 00219290. doi: 10.1016/j.jbiomech.2009.12.012. URL http://linkinghub.elsevier.com/retrieve/pii/S0021929009007210.

F. C. Anderson and M. G. Pandy. Dynamic optimization of human walking. *Journal of biomechanical engineering*, 123(5):381–390, 2001. ISSN 0148-0731.

S. Annunziata and A. Schneider. Physiologically based control laws featuring antagonistic muscle co-activation for stable compliant joint drives. *Applied Bionics and Biomechanics*, 9:249–266, 2012. ISSN 11762322. doi: 10.3233/ABB-2012-0062. URL /citations?view{_}op=view{_}citation{&}continue=/scholar?hl=de{&}start=50{&}as{_}sdt=0,5{&}scilib=1{&}scioq=dcmd+locust+gabbiani{&}citilm=1{&}citation{_}for{_}view=sUrzMWQAAAAJ:l7t{_}Zn2s7bgC{&}hl=de{&}oi=p.

R. Cham and M. S. Redfern. Changes in gait when anticipating slippery floors. *Gait and Posture*, 15:159–171, 2002. ISSN 09666362. doi: 10.1016/S0966-6362(01)00150-3.

J. S. Coggan, J. Grutzendler, D. L. Bishop, M. R. Cook, W. Gan, J. Heym, and J. W. Lichtman. Age-associated synapse elimination in mouse parasympathetic ganglia. *Journal of Neurobiology*, 60(2):214–226, 2004. ISSN 00223034. doi: 10.1002/neu.20022.

M. De Lasa and A. Hertzmann. Prioritized optimization for task-space control. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, 1(2):5755–5762, 2009. doi: 10.1109/IROS.2009.5354341.

E. De Vlugt, A. C. Schouten, and F. C. T. Van Der Helm. Adaptation of reflexive feedback during arm posture to different environments. *Biological Cybernetics*, 87(1):10–26, 2002. ISSN 03401200. doi: 10.1007/s00422-002-0311-8.

S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, C. T. John, E. Guendelman, and D. G. Thelen. OpenSim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950, 2007. ISSN 00189294. doi: 10.1109/TBME.2007.901024.

V. Dietz. Proprioception and locomotor disorders. *Nature Reviews Neuroscience*, 3(10):781–790, oct 2002. ISSN 1471-003X. doi: 10.1038/nrn939. URL http://www.nature.com/doifinder/10.1038/nrn939.

T. W. Dorn, J. M. Wang, J. L. Hicks, and S. L. Delp. Predictive Simulation Generates Human Adaptations during Loaded and Inclined Walking. *Plos One*, 10(4):e0121407, 2015. ISSN 1932-6203. doi: 10.1371/journal.pone.0121407. URL http://dx.plos.org/10.1371/journal.pone.0121407.

J. Duysens and Van de Crommert HW. Neural control of locomotion; The central pattern generator from cats to humans. *Gait & posture*, 7(2):131–141, 1998. ISSN 1879-2219. doi: 10.1016/S0966-6362(97)00042-8. URL http://www.sciencedirect.com/science/article/pii/S0966636297000428$\delimiter"026E30F$nhttp://www.ncbi.nlm.nih.gov/pubmed/10200383.

F. Dzeladini, J. van den Kieboom, and A. Ijspeert. The contribution of a central pattern generator in a reflex-based neuromuscular model. *Frontiers in Human Neuroscience*, 8 (June):1–18, 2014. ISSN 1662-5161. doi: 10.3389/fnhum.2014.00371. URL http://www.frontiersin.org/Human{_}Neuroscience/10.3389/fnhum.2014.00371/abstract.

P. A. Forbes, R. Happee, F. C. van der Helm, and A. C. Schouten. EMG feedback tasks reduce reflexive stiffness during force and position perturbations. *Experimental Brain Research*, 213 (1):49–61, 2011. ISSN 00144819. doi: 10.1007/s00221-011-2776-y. URL http://www.ncbi.nlm.nih.gov/pubmed/21717098.

S. S. Geertsen, K. Stecina, C. F. Meehan, J. B. Nielsen, and H. Hultborn. Reciprocal Ia inhibition contributes to motoneuronal hyperpolarisation during the inactive phase of locomotion and scratching in the cat. *The Journal of Physiology*, 589(1):119–134, 2011. ISSN 0022-3751. doi: 10.1113/jphysiol.2010.199125. URL http://www.jphysiol.org/cgi/doi/10.1113/jphysiol.2010.199125.

T. Geijtenbeek, M. van de Panne, and a. F. van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6):1–11, nov 2013. ISSN 07300301. doi: 10.1145/2508363.2508399. URL http://dl.acm.org/citation.cfm?doid=2508363.2508399.

H. Geyer and H. Herr. A Muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18:263–273, 2010. ISSN 15344320. doi: 10.1109/TNSRE.2010.2047592.

H. Geyer, a. Seyfarth, and R. Blickhan. Positive force feedback in bouncing gaits? *Proceedings of the Royal Society B: Biological Sciences*, 270(1529):2173–2183, 2003. ISSN 0962-8452. doi: 10.1098/rspb.2003.2454. URL http://rspb.royalsocietypublishing.org/cgi/doi/10.1098/rspb.2003.2454.

N. Hansen. The CMA evolution strategy: A Comparing Review. *Vu le*, 102(2006):1–34, 2006. URL http://www.lri.fr/{~}hansen/cmatutorial110628.pdf.

N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1):1–18, 2003. ISSN 1063-6560. doi: 10.1162/106365603321828970.

A. V. Hill and Se. The heat of shortening and the dynamic constants of muscle. *Biophysics, Department of Physiology, University College, London*, 1938.

K. H. Hunt and F. R. E. Crossley. Coefficient of Restitution Interpreted as Damping in Vibroimpact. *Journal of Applied Mechanics*, 42(2):440, 1975. ISSN 00218936. doi: 10.1115/1.3423596. URL http://appliedmechanics.asmedigitalcollection.asme.org/article.aspx?articleid=1402467.

J. D. Kralik, D. F. Dimitrov, D. J. Krupa, D. B. Katz, D. Cohen, and M. A. Nicolelis. Techniques for Chronic, Multisite Neuronal Ensemble Recordings in Behaving Animals. *Methods*, 25(2):121–150, 2001. ISSN 10462023. doi: 10.1006/meth.2001.1231. URL http://linkinghub.elsevier.com/retrieve/pii/S1046202301912319.

S. Künzell, C. Augste, M. Hering, S. Maier, A.-M. Meinzinger, and D. Sießmeir. Optimal control in the critical phase of movement: a functional approach to motor planning processes. *Acta psychologica*, 143(3):310–6, 2013. ISSN 1873-6297. doi: 10.1016/j.actpsy.2013.04.013. URL http://www.ncbi.nlm.nih.gov/pubmed/23727597.

D. Lobo and M. Levin. Inferring Regulatory Networks from Experimental Morphological Phenotypes: A Computational Method Reverse-Engineers Planarian Regeneration. *PLOS Computational Biology*, 11(6):e1004295, 2015. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1004295. URL http://dx.plos.org/10.1371/journal.pcbi.1004295.

M. Millard, T. Uchida, A. Seth, and S. L. Delp. Flexing Computational Muscle: Modeling and Simulation of Musculotendon Dynamics. *Journal of Biomechanical Engineering*, 135(2):021005, 2013. ISSN 0148-0731. doi: 10.1115/1.4023390. URL http://biomechanical.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4023390.

E. K. Miller and M. a. Wilson. All My Circuits: Using Multiple Electrodes to Understand Functioning Neural Networks. *Neuron*, 60(3):483–488, 2008. ISSN 08966273. doi: 10.1016/j.neuron.2008.10.033. URL http://dx.doi.org/10.1016/j.neuron.2008.10.033.

W. Mugge, D. A. Abbink, A. C. Schouten, J. P. A. Dewald, and F. C. T. van der Helm. A rigorous model of reflex function indicates that position and force feedback are flexibly tuned to position and force tasks. *Experimental Brain Research*, 200(3-4):325–340, 2010. ISSN 0014-4819. doi: 10.1007/s00221-009-1985-0. URL http://link.springer.com/10.1007/s00221-009-1985-0.

F. Pedregosa and G. Varoquaux. Scikit-learn: Machine Learning in Python. *Journal of Machine ...*, 12:2825–2830, 2011. URL http://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf.

U. Proske and S. C. Gandevia. The Proprioceptive Senses: Their Roles in Signaling Body Shape, Body Position and Movement, and Muscle Force. *Physiological Reviews*, 92(4):1651–1697, 2012. ISSN 0031-9333. doi: 10.1152/physrev.00048.2011.

M. a. Sherman, A. Seth, and S. L. Delp. Simbody: Multibody dynamics for biomedical research. *Procedia IUTAM*, 2:241–261, 2011. ISSN 22109838. doi: 10.1016/j.piutam.2011.04.023. URL http://dx.doi.org/10.1016/j.piutam.2011.04.023.

T. Sinkjaer, J. B. Andersen, M. Ladouceur, L. O. Christensen, and J. B. Nielsen. Major role for sensory feedback in soleus EMG activity in the stance phase of walking in man. *The Journal of physiology*, 523 Pt 3:817–827, 2000. ISSN 0022-3751. doi: 10.1111/j.1469-7793.2000.00817.x.

S. Song and H. Geyer. Regulating speed and generating large speed transitions in a neuromuscular human walking model. *Pro[1] S. Song and H. Geyer, "Regulating speed and generating large speed transitions in a neuromuscular human walking model," Proc. - IEEE Int. Conf. Robot. Autom., pp. 511–516, 2012.ceedings - IEEE International Conference on Robotics and Automation*, pages 511–516, 2012. ISSN 10504729. doi: 10.1109/ICRA.2012.6225307.

S. Song and H. Geyer. A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion. *The Journal of Physiology*, 593(16):3493–3511, 2015. ISSN 00223751. doi: 10.1113/JP270228. URL `http://doi.wiley.com/10.1113/JP270228`.

C. D. Takahashi, R. a. Scheidt, and D. J. Reinkensmeyer. Impedance control and internal model formation when reaching in a randomly varying dynamical environment. *Journal of neurophysiology*, 86(2):1047–1051, 2001. ISSN 0022-3077. doi: citeulike-article-id:761586.

J. C. Tapia, J. D. Wylie, N. Kasthuri, K. J. Hayworth, R. Schalek, D. R. Berger, C. Guatimosim, H. S. Seung, and J. W. Lichtman. Pervasive synaptic branch removal in the mammalian neuromuscular system at birth. *Neuron*, 74(5):816–29, 2012. ISSN 1097-4199. doi: 10.1016/j.neuron.2012.04.017. URL `http://www.sciencedirect.com/science/article/pii/S0896627312003856`.

D. G. Thelen, F. C. Anderson, and S. L. Delp. Generating dynamic simulations of movement using computed muscle control. *Journal of Biomechanics*, 36:321–328, 2003. ISSN 00219290. doi: 10.1016/S0021-9290(02)00432-3.

N. Van der Noot, A. J. Ijspeert, and R. Ronsse. Biped gait controller for large speed variations, combining reflexes and a central pattern generator in a neuromuscular model. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6267–6274, 2015. doi: 10.1109/ICRA.2015.7140079. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7140079`.

M. J. Wagner and M. a. Smith. Shared internal models for feedforward and feedback control. *The Journal of neuroscience*, 28(42):10663–73, 2008. ISSN 1529-2401. doi: 10.1523/JNEUROSCI.5479-07.2008. URL `http://www.ncbi.nlm.nih.gov/pubmed/18923042`.

J. Wang, S. Hamner, and S. Delp. Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives. *ACM Transactions on Graphics*, 31(4), 2012. URL `http://www.stanford.edu/group/nmbl/publications/pdf/Wang2012.pdf`.

Appendix **A**

PredictiveSim Documentation

This is documentation for *JWPredictiveSim*, which is a derivative of *PredictiveSim* by Dorn et al. ((2015)). The following sections extend upon the original documentation while introducing a couple of new features from *JWPredictiveSim*.

A.1 Getting Started With Predictive Sim

A.1.1 Reaching First Simulation

PredictiveSim is a package released in 2015 alongside the publication: Predictive Simulation Generates Human Adaptations during Loaded and Inclined Walking by Dorn et al. ((2015)). The system comes as a bundle of source files: 9 c++ files, 9 header files, a CMakeLists file, 2 Matlab methods for generating initialization files, a list of OpenSim model files and a README. The README is the only source of documentation provided including 3 example commands with explenations of each as well as the following guide for getting it running:

"Run cmake to build the executable, specify the installation location for OpenSim. This source code has most recently been tested using OpenSim 3.2, Simbody 3.3.1, and gcc-4.9 on OSX Yosemite (10.10.3).
Some auxiliary files need to be generated by running createOptInitFiles.m using either Matlab or Octave."

A.1.2 Compiling on Windows

Despite the verified compatibility with OSX, the system was first sought compiled on Windows for reasons of better prior understanding of platform and available tools. Microsoft Visual Studio 2015 (VS2015) was chosen as the Windows compiler as it was the latest verified compiler for the two dependencies: OpenSim and Simbody. Simbody and Opensim both have precompiled packages available for Windows which was installed.

MPICH2 or OpenMPI for windows?!

The PredictiveSim CMake file was then setup pointing to their installation directories, using VS2015 as the compiler and enabling MPI. It was found that PredictiveSim crashes without

proper error handling if run without MPI or using only 1 node. Setting up all linked libraries in VS2015 required some meddling around but everything compiled when it was finally sorted.
PredictiveSim requires a list of input files when executed. By default these are assumed to be located in the parent directory of the executable. The required files are the auxiliary files generated by $createOptInitFiles.m$: $optimization_values.sto$, $optimization_activeparams.sto$, $optimization_bounds.sto$ and $optimization_ranges.sto$ and a model file (By default $Humanoid2D.osim$ is used). Further inputs include a control file using the argument $-cf$, $-viz$ enabling the visualizer, $-t$ setting the length of the simulation. A complete list of arguments can be found in section

With the above in place, the bundled control file *normalwalk.sto* can then be simulated for 10 seconds and visualized by the following command.:

```
pdsim.exe -m ../Humanoid2D.osim -cf ../normalwalk.sto -t 10 -viz
```

An optimization can then be launched by the following command. Here the target speed $1.5^{m}/_{s}$, step size for the optimizer is 0.005, the optimization uses 10 seconds of simulation per evaulation and it has been launched on 50 nodes using MPI.

```
mpiexec -n 50 pdsim.exe -m ../Humanoid2D.osim -opt 0.005 -target 1.5 -t 10
```

A.2 Procedure of use

This procedure assumes that *PredictiveSim* has been correctly compiled along with MPI, producing a pdsim executable as well as containing the correct files it its parent directory (See Appendix A). To initiate the system 4 auxiliary files are required. These are generated by a supplied matlab script, $createOptInitFiles.m$, which creates: $optimization_activeparams.sto$, $optimization_values.sto$, $optimization_ranges.sto$ and $optimization_bounds.sto$. The matlab file contains a list of all input parameters along with their corresponding values, range, bounds and whether or not they should be optimized. When the script is run, the auxiliary files each contain their corresponding values in a format that OpenSim can read. For performing an optimization on Windows the following command can be used:

```
mpiexec -n 50 pdsim.exe -target 1.5 -t 10
```

This initiates 50 parallel nodes through MPI, running the a pdsim optimization for 10 seconds aiming for a target velocity of 1.5 m/s. While the optimization procedure is running iteratively it will write a control file named $result_itr\#.sto$ each time a better fitness value is found, where the # is the iteration number. This file contains a list of control parameters that when simulated gives the latest and greatest fitness. When a sufficiently good fitness is found the process can be stopped and the resulting control file can be simulated with:

```
pdsim.exe -cf result_itr#.sto -t 10 -viz
```

This will perform a 10 seconds simulation run through using the parameters in the control file $results_itr\#.sto$ while writing out a file called $\_results.sto$ that contain all data about limb position, velocities, muscle forces, ground forces etc. for each time step. The parameters in

_results.sto can then be plotted through OpenSim or any other tool capable of extracting the information from the file, or the motion can be replayed using the replay argument:

```
pdsim.exe -replay _results.sto -t 10 -viz
```

An optimization using PredictiveSim will output a number of files. These include a *result_opt_progress.csv* file that contains 4 columns: iteration number, fitness value, current best fitness and time in seconds. For each new iteration a new line will be added to this file. Secondly a number of *result_itr####.sto* files will be created for each improved fitness. These are control files containing all parameter values required to reach the given fitness.To perform an optimization a number of inputs are required. For initiating an optimization the initial conditions are by default given by the 4 auxiliary files produced by the attached Matlab script. These specify all initial values, which parameters are to be optimized as well as defining the bounds for each parameter. Additionally there is an option to use a specific control file as initial condition. Doing this will overwrite the values stated in the auxiliary files by the ones given by the control file. The above mentioned *result_itr####.sto* files can be used for this. More details about input and output files in Appendix A.5.

A.2.1 Summary

Software Used

- Windows: Microsoft Visual Studio 2015 Pro
- Linux/OSX: gcc 4.9.2
- CMake 3.2.2
- Simbody 3.5.3
- OpenSim 3.3
- OpenMPI or MPICH2 for windows?

Notes

- Code requires MPI to run. Will crash without it
- CMakeList does not include Window/Linux paths by default
- (linux) Setup CMake with:

$$BUILD\_USING\_OTHER\_LAPACK =: liblapack.so.3; libblas.so.3$$

A.3 MPI and Cluster Simulation

- MPCH2
- Suggested setup

A.4 CMA-ES

1. stepSize (σ) - progression. It should lower along with iterations?
2. individual (x)

3. parents (μ)
4. population size / offspring (λ)
5. damping?
6. standard deviation (σ_0)
7. Rank?

A.5 Input/Output data

Input Data

1. model file,
2. control file. This is includes the 4 files generated by the matlab script as well as an optional file for defining other initial parameters, added by the argument "-cf"
3. target speed
4. time

A.6 Details

A.6.1 Optimized Parameters

There are by default 70 parameters that are being optimized, with 77 parameters in total where the last 7 describe trunk translation on the x-axis as well as 6 static parameters describing the contact forces: *stiffness*, *dissipation*, *static_friction*, *dynamic_friction*, *viscous_friction* and *transition_velocity*. What are the _p and _q parameters? - Containers for muscle previous muscle activation. _p in stance phase and _q in swing phase.

```
'trunk_extension',
%'trunk_tx',
'trunk_ty',
'hip_r_flexion',
'knee_r_extension',
'ankle_r_dorsiflexion',
'hip_l_flexion',
'knee_l_extension',
'ankle_l_dorsiflexion',
'trunk_extension_u',
'trunk_tx_u',
'trunk_ty_u',
'hip_r_flexion_u',
'knee_r_extension_u',
'ankle_r_dorsiflexion_u',
'hip_l_flexion_u',
'knee_l_extension_u',
'ankle_l_dorsiflexion_u',
%'stiffness',
```

```
%'dissipation',
%'static_friction',
%'dynamic_friction',
%'viscous_friction',
%'transition_velocity',
'GMAX_p',
'ILPSO_p',
'HAMS_p',
'RF_p',
'VAS_p',
'GAS_p',
'SOL_p',
'TA_p',
'GMAX_q',
'ILPSO_q',
'HAMS_q',
'RF_q',
'VAS_q',
'GAS_q',
'SOL_q',
'TA_q',
'G_sol',
'G_ta',
'l_off_ta',
'G_solta',
'G_gas',
'G_vas',
'k_phi',
'phi_k_off',
'k_p_glu',
'theta_ref',
'k_d_glu',
'Delta_S_glu',
'G_ham',
'G_glu',
'G_hfl',
'l_off_hfl',
'G_hamhfl',
'l_off_ham',
'k_lean',
'k_p_hfl',
'k_d_hfl',
'k_p_ham',
'k_d_ham',
'Delta_S_hfl',
'Delta_S_rf',
'Delta_S_vas',
'k_p_glu_sp',
'k_d_glu_sp',
'k_p_hfl_sp',
'k_d_hfl_sp',
'k_p_vas_sp',
'k_d_vas_sp',
```

```
'htheta_ref_sp',
'phi_ref_sp',
'sp_threshold',
'simbicon_cd',
'simbicon_cv'
```

A.6.2 Delays

0 delay for reading COM position 0.01s delay for reading

A.6.3 Source Files

- main.cpp Main loop, checking commandline argumnets, initiating system, loading files and starting either the optimizer or a single simulation with or without visualization.
- GeyerHerrController.cpp
  Computing muscle controls and initiating controls with start state. This is where all reflexes are modelled and connected.
- cmaes.c and cmaes_interface.h
  The CMAES algorithm, as described by Hansen et al. ((2003))
- CMAOptimizer.cpp
- EventHandlers.cpp Simulation State Switching using events from foot contacts and visualization key triggers
- PredictiveOptimizationSystem.cpp Objective Function
- SimpleMuscle.cpp Defining the muscles
- SoftCoordinateLimitForce.cpp Defining the ground reaction forces
- Utils.cpp A list of utilities including conversion functions from standard parameter input to a normalized optimizer input and back again.

Appendix B

Supplementary Sections

B.1 Tools Details

B.1.1 Machine Learning

The Machine Learning (ML) library used here is the Python SKLearn library ((Pedregosa and Varoquaux, 2011)). Machine Learning refers in this project to linear regression models. These are used as a method for automatically looking for patterns in large and multidimensional dataset. First an ML algorithm is trained and then it is validated. The validation happens by separating the dataset into training and validation data. The algorithm is then trained by the training set and subsequently asked to predict the validation set. For quantifying this the library includes a 'regr.score' function that output the corresponding R^2 value for a series of validations.

B.1.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is here applied through the Python based SKLearn library Pedregosa and Varoquaux ((2011)). The basical functionality of PCA is to perform a coordinate transformation to a dataset where the first axis of that new coordinate system is pointing in the direction where the dataset is the widest. This means that PCA looks for the directions where the data varies the most and aligns its axis according to these direction. Each axis is defined by a weighting of each of the parameters in the dataset and their Explained Variance Ration (EVR) is given, which states how much of the variance in the dataset is explained by each axis.

B.2 Software and Additional Features

B.2.1 Approaching the Software

The software (*PredictiveSim*) came with an initial README describing the basics for getting started (See Appendix C). For initial testing Visual Studio was used for compiling on Windows, using CMake and MPICH2. Windows was also used as the primary platform for further development of the software. For running optimizations on DTU's High Performance Computing

cluster (HPC), both OpenSim ((Delp et al., 2007)) and SimBody ((Sherman et al., 2011)) needed to be compiled as well due to a Linux-based platform (More info can be found in Appendix A) With the software running the code was explored and annotated.

Preparing *PredictiveSim* for Analysis

PredictiveSim produces a list of files when run, which can be used for analysis (See Appendix A.6 for details). However, in order to accommodate a more in-depth analysis it was found that more details would be needed as well as a new approach to save out and handle a large amount of simulation data. This both raised the requirement for data management as well as labelling so that processes could be automated and that all information for each simulation could be easily related. This was achieved in part by adding metadata to all files produces as well as extending some of the basic functionality of the software. The metadata would then include all information about achieved fitness value, execution parameters, file dependencies etc. In conjunction with this a number of python tools were made to automate most of the repeating logical tasks, extract information from large number of files, plot and generate new files. Beyond that a number of small additions were made to the software allowing among other to stop optimization automatically when a certain convergence threshold had been reached as well as enabling saving out parameters for all simulations during an optimization process. Saving out everything would then save a $sim_dump.csv$ file with each line containing all simulated parameters and their corresponding fitness. This allowed easy load into python using the Pandas library. Further information can be found in Appendix B.2

B.2.2 Convergence Check

After analyzing the development of parameters Defined a limit of 30 consecutive iterations without improvements which would cause the optimization procedure to exits automatically

B.2.3 Adding metadata

To make sure that no data was ever jumbled or confused as well as enabling automatic and easy retrieval of a range of extra data, a header was added to each $result_itr\#.sto$ file containing the following meta data:

- execFile: Filename of executable
- execCmd: Commandline arguments used
- finaltime: Simulation time
- targetVel: Self explanatory
- backpackLoad: Self explanatory
- visualize: Self explanatory
- printDetailsToFile: Self explanatory
- doOpt: Self explanatory
- resumeIndex: Self explanatory
- stepSize: Self explanatory
- replayFile: Self explanatory
- modelFile: Self explanatory
- ctrlFile: Self explanatory
- ctrlPrefix: Self explanatory

- fitness: Fitness value using the current file
- noImprovLimit: How many iterations are allowed to pass before exit
- numtasks: How many cores used
- pertAmp: Perturbation Amplitude
- secondsPassedTotal: Elapsed time of whole optimization until this point
- secondsPassedItr: Elapsed time of this particular iteration
- VERSION_NAME: Self explanatory
- VERSION_NUMBER: Self explanatory

A timestamp for each iteration was also added to the *result_opt_progress.csv* file as well as timestamps in the Console Out statements for receiving data from each MPI node. These two additions allowed for easy monitoring of the progression of each optimization.

B.2.4 Added a list of input arguments for easier testing and accessing new features

- -ctrlPrefix <string>: Cmdln specification of control prefix, allowing to quickly switch between several files
- -pertAmp <amp>: Define Perturbation Amplitude, disabled if not set.
- -version: Printing current software version name and number and quits
- -supraCtrl: Enabling supraspinal control through currently hardcoded lists
- -loadCtrls: Loading previsouly saved state as the current initial state (Not implemented fully)
- -saveCtrls <time in sec>: Save out complete state of system at specified time (Not yet fully implemented)
- -saveAllSim: Saves out parameters and fitness from all inner simulations to a csv file.
- -brf <maxiteration>: Uses Blind Random Search for optimizing instead of CMA-ES

remember to update here

B.2.5 Writing the Basic Tools

During this project most recurring logical tasks were automated using python tools. These include simulation monitoring, extracting and converting data, sorting through large amount of files with certain values, generating new control files with simple math operations and plotting of arbitrary data. All tools produced during this project are maintained in a git repository which can be found on
`https://bitbucket.org/jakobwelner/pdsim-pythonutils`
Tools were all written in Python, using a list of libraries including Matplotlib, Pandas and sklearn ((Pedregosa and Varoquaux, 2011)).

Monitoring Some monitoring tools were made to run remote on the computing cluster, where simple arguments could traverse through all optimizations and printing out their current status, latest fitness value, average iteration time and optimization health. The health value was based on an arbitrary algorithm based on the convergence limit as described in section B.2.2. This number would describe the sum of squared difference to the current best fitness, looking 30 iterations back, thus reaching 0 when fitness is stable for 30 iterations, ie. no improvement was made.

Management A list of python scripts were made to handle data files including reading/writing of OpenSim Storage files (.sto extension) including both values and meta data. Other tools were made to traverse file structures containing optimization data while extracting all the best iterations from each optimization while grouping and naming the files accordingly.

Plotting Using Pandas and Matplotlib a few iPython notebooks were set up to quickly visualize any optimization data, compare it to other sample and combine multiple samples using simple mathematical expressions.

Analysis Python was also used for performing simple data analysis using the python machine learning library SKLearn ((Pedregosa and Varoquaux, 2011)). This allowed for training machine learning algorithms on the data and performing Principal Component Analysis (PCA) for determining parameter weighting.

3D-Visualization To visualize the animations a tool was made to extract model-parameters from the Humanoid2D.osim file containing the OpenSim model file with dimensions, pivots and muscle attachments. This data was used to reconstruct the model in Alias Maya^{tm} allowing for more powerful modelling, rendering and animation tools.

Appendix C

PredictiveSim Original Readme

```
Quick guide to try out the code:

Run cmake to build the executable, specify the installation location for
OpenSim. This source code has most recently been tested using OpenSim 3.2,
Simbody 3.3.1, and gcc-4.9 on OSX Yosemite (10.10.3).

Some auxiliary files need to be generated by running createOptInitFiles.m
using either Matlab or Octave.

================================================================================
To visualize the default control parameters using model Humanoid2D.osim for
10 seconds:
./pdsim -m ../Humanoid2D.osim -t 10 -viz

================================================================================
To visualize the example controller normalwalk.sto using the model Humanoid2D.osim for 10 seconds:
./pdsim -m ../Humanoid2D.osim -cf ../normalwalk.sto -t 10 -viz

================================================================================
To optimize for flat ground walking at 1.5 m/s using CMA and MPI:
mpiexec -n 50 ./pdsim -m ../Humanoid2D.osim -opt 0.005 -target 1.5 -t 10

The above command spawns 50 processes in parallel, the code uses 50-1=49 CMA
samples per iteration by default. The initial CMA stepsize is set to 0.005.

A controller solution is saved (result_itrXX.sto) when the current best value
is improved at a given iteration. These files can be visualized using the
-cf flag.

The .osim file describes the problem environment. For example,
-m Humanoid2DUphill10deg.osim simulates walking up a 10 degree incline.
The initialization for CMA can be modified by specifying a controller using
the -cf flag. For example, using -cf normalwalk.sto and -m
Humanoid2DUphill10deg.osim seeds the 10 degree incline optimization using the
walking controller optimized for flat ground.
```

```
================================================================================

In practice, you might need to run MPI across multiple machines to handle
the amount of parallel processing necessary. You will have to consult your
own system admin to set this up.
```