Modular Engineering of Neuromuscular Gait Simulators

by

Matthew D. Furtney

B.S., Biomedical Engineering and Computer Science
University of Miami (2013)

Submitted to the Program of Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Signature of Author: Signature redacted
Program in Media Arts and Sciences
May 6, 2016

Certified by: Signature redacted
Hugh Herr, Ph.D.
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by: Signature redacted
Pattie Maes, Ph.D.
Academic Head
Program in Media Arts and Sciences

Modular Engineering of Neuromuscular Gait Simulators

by
Matthew D. Furtney

Submitted to the Program of Media Arts and Sciences,
School of Architecture and Planning, on May 6, 2016
in partial fulfillment of the requirements for the degree of
Master of Science

Abstract

In this thesis we present a novel approach to the computer simulation of forward dynamic gait models and the optimization of parameters that must be tuned for such models. This methodology is not limited to gait simulation, and could be useful for any situation in which a complex Simulink model requires variables to be tuned via machine learning to optimize all heuristic that can only be evaluated via simulation.

Through the lens of Biomechatronic engineering research, we combine the fundamentals of software engineering with a refinement of the best practices of Matlab and Simulink programming and a working knowledge of inherent Matlab and Simulink constraints to construct a framework for rapid model development.

Key features of this methodology include: the use of Simulink as an environment for rapidly prototyped models, the use of and construction of custom Simulink libraries, and use of the Matlab Optimization Toolbox. This methodology uses parallel evaluation of rapid acceleration Simulink executables to minimize optimization time, and allow research teams to take advantage of parallel processing and cloud computing.

This methodology was applied to a bouncing gait model developed by Hartmut Geyer for evaluation. We demonstrate its effectiveness by simulating this model using a custom library of model components, such as ground contact model, Stateflow control, heuristic computation, and body segments.

Thesis Supervisor: Hugh Herr, Ph.D.

Title: Associate Professor of Media Arts and Sciences

Modular Engineering of Neuromuscular Gait Simulators

by

Matthew D. Furtney

The following served as readers on the thesis committee:

Research Advisor Signature redacted

Hugh Herr, Ph.D.
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Thesis Supervisor: Signature redacted

Ed Boyden, Ph.D.
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Thesis Supervisor: Signature redacted

Brian Umberger, Ph.D.
Associate Professor, Department of Kinesiology
University of Massachusetts, Amherst

Acknowledgments

Without these amazing people, I would not have finished.

Hugh Herr	Inspirational Advisor
Sarah Rowlinson	Motivator, Supporter
Rick and Barbara Furtney	Parents and Engineering Advisors
Matt Chun	Ideas Facilitator and Exceptional UROP
Dalia Leibowitz	Voice of Reason and Friend
Ken Endo	The Shoulders On Which I Stand
Jiun-Yih Kuan	Best Office Mate
David Hill	Also Best Office Mate
Shuo Wang	Best Locked-Out-of-Office Mate
Michael Eilenberg	Friend and Answerer of a Thousand Questions
Brian Umberger	Reader
Ed Boyden	Reader

Contents

List of Figures

Preface

At the frontier of gait modeling, a multitude of models that describe different gait exist. Some of these models are quite complex, in that they involve defining hundreds of parameters. All too often, these models are simulated using custom software that has a lifespan following its creator's PhD timeline. Development time is spent re-implementing the same model components over and over.

When I started my research project, I set a goal of taking a specific model, and improving it so that it could walk at a broad range of speeds with human like kinematics and metabolism. And while the most academically relevant parts of the model, its controls, were simple, the full implementation of the model was overwhelming complex. I started with a complex implementation, and I only made it more complex with my extra layer of modifications (take a look at page 50). By the end, I had something that would completely overwhelm any new collaborator. I had marooned myself on an island of complexity. It turned out that building a forward dynamic gait simulator was by its nature, a complex task. I realized that the biggest impediment to new gait models was not creativity or biological insight. It was the development cycle of the simulator.

I took this challenge as guidance for a new direction in my research. Instead of working on one improvement to one model, I should improve the weak link in the research cycle. There was a clear need for a forward dynamics simulation tool that could make iterative gait model design and validation as fast as it could be.

Introduction

Motivation

For millennia, humans have been augmenting their mobility. From the domestication of horses to the invention of flight, the speed and efficiency of human locomotion has increased dramatically. With large machines, such as trucks, trains and ships, human locomotion is now markedly more efficient than anything found in biology in terms of energy cost of transport [1], [2].

Despite all of this progress in transportation, technology still struggles with augmenting and properly restoring one of the most important forms of human locomotion: walking [3]. Walking is highly efficient for short-range transportation, and is a useful tool for maintaining wellness. After decades of research, recent studies have shown that by augmenting walking, we can enable people to carry heavier loads with less energy expenditure [3]. Walking augmentation exoskeletons hold the promise of increasing the mobility of people with muscle atrophy, peripheral nerve disorders, obesity, or arthritis.

Restoration and augmentation of human walking is a hard problem. Since General Electric made an initial foray into exoskeleton design with the Hardiman I project, it has taken the field over 40 years of research to design an autonomous augmenting exoskeleton that can make the seemingly simple act of walking with a pack more metabolically efficient [3], [4]. Even the most advanced below knee prosthetic legs would benefit from improved stability [5], and above knee prosthesis systems have significant deficits. Users with the best foot to femur systems on the market still have 65% higher COT during walking and slower preferred walking speed [6]. The asymmetry

in the gait of unilateral above knee amputees is noticeable to the untrained eye. There have been several hurdles holding back progress in this field.

The first hurdle is cost. Making one prototype to test one theory has been the status quo. That prototype is then re-designed to test the next theory, or it is abandoned completely. This would be fine except prototypes are expensive to build and expensive to test. Each design requires many hours of cad design and custom fabrication time. In addition, experiments done on human subjects have a high cost per hour.

The second hurdle is safety. Exoskeletons must be safe to test on human subjects, yet they must exert torques that rival or exceed human capability. Amputees testing prosthetic legs are at risk of falling if a device acts unexpectedly or fails. Designing for safety increases cost and time to the prototype design process.

The final hurdle is time. Since these prototypes and the equipment used to test them are so expensive, there is usually only one instance of any given prototype. Testing a large sample size must be done in series.

These three hurdles explain the demand for using gait models to aid in device design. An accurate dynamic model of human walking and associated neural reflexes can be used for virtually testing new designs for exoskeletons, prosthetic legs, and the systems that control them. In addition, the model helps us better understand the way the muscles and reflexes normally function. In the virtual world, the cost per digital prototype is lower, and there are no safety concerns. The only limit is human creativity and the number of processors available. For example, instead of testing one design on 80 live individuals, create a database of the recorded

walking kinematics, metabolic rate, and mass distribution of 80 individuals, and then test an unlimited number of exoskeleton or prosthetic designs on those modeled individuals.

Currently, many dynamic models of walking exist. Some emulate the kinematics of test subjects at multiple speeds, and others are able to simulate muscle function with reasonable metabolic cost. But so far, there is still a need for models with more biological simulation accuracy [7]. Having a model that can more accurately predict the metabolic cost of gait is a useful tool for the design process of an exoskeleton or prosthetic leg. Improvements in gait modeling technology can lead to more natural gait in amputees with powered prosthetic legs, and it can lead to more efficient exoskeletons that move in harmony with the wearer, providing assistance at the most impactful points of the gait cycle.

Computational Challenge

The biggest computational cost in creating and testing gait models is the search for properly tuned input parameters. Parameters such as tendon stiffness, maximum muscle force, and controller gains and timings must be precisely tuned to produce the desired gait. Without careful tuning, the simulated walker just falls over or produces erratic behavior. Most of these tuned variables have complex interactions that can only be determined by simulation. Thus it is necessary to run the simulation thousands of times with different sets of inputs as part of a machine learning objective function to optimize the input parameters. Using a genetic algorithm with a population of 500 and 100 generations requires 50,000 simulations. Even worse, the more tuned parameters a model has, the larger the population one will need to find good solutions.

If it takes 10 minutes to evaluate a simulation, the above example takes almost one year to search for an optimized set of input parameters! Even cutting things down to 30 second simulation time

results in a 17 day computation. This sort of time is expensive in terms of computers, and it is impractical wait for weeks just to see if a small change to the model can emulate the desired gait at all, let alone closely match clinical data.

The critical challenge is this: as a developer, I need to frequently test my model and any changes that I make. But any substantial change to the model necessitates the re-tuning of input parameters in order to run a simulation that emulates the desired gait. Most scientifically interesting changes, such as changing your tendon spring equation, are substantial enough to necessitate the re-tuning (optimization) of input parameters. I might wait weeks just to see a simulation result that falls over, and that a change to the model has a bug in its implementation. Building an array of tools to test changes would be a reasonable step, but the fact of the matter is that developing these tools is in itself expensive. Each component demands a new tool. If we can reduce the optimization time down to one day or less, the developer can test new ideas and iterate rapidly, without wasting time creating complex debugging tools.

So we came up with a new project: design a method that improves the efficiency of the model evaluation process, and thereby accelerate the development of future gait models.

Requirements

The following requirements define the project:

1. The simulation shall be capable of closed loop feedback control.
2. The optimization shall evaluate multiple simulations in parallel to reduce the time the developer spends waiting for the optimization to complete.
3. The model shall be implemented in an organized and modular way. Changing one part of the model, such as which muscle model is used, shall require only one implementation of

the new muscle model, and a simple substitution of which library components are used for each relevant muscle in the model.

4. The main body of code that optimizes the model parameters shall be simple, well organized, and reusable across different models.
5. There should be an accumulating library of reusable model components, such as body segments, muscles, ground contact models, and controllers. Each time a model is implemented, this library is increased in size. Each time a library component is improved in efficiency or organization, the change is persistent and spreads to others who are using the same library.
6. There shall be an accumulating archive of successfully running models that can used as a starting place for building new models, so that each new component can be tested within a whole system.
7. The code used for the optimization shall be sourced from a library with the choice of machine learning algorithm being as simple as changing a few lines of code. Implementing a custom genetic algorithm shall be unnecessary.
8. There shall be a highly customizable analysis mode that can run simulations using pre-tuned parameter sets and produce graphic videos of the gait, and any desired plot of simulation state information like force, torque, etc.

With these requirements in mind, we conducted a survey of existing tools and found that none met our requirements. It was necessary to make a new tool.

Model

In this paper, the term 'model', is used to describe a set of rules and equations that represent a phenomenon. In this paper, the models we discuss will be models that describe gait by defining

equations and relationships of the human body and its environment. The model includes equations for controlling neural reflexes and delays, the way muscles function, and the geometries of body segments, the biological muscle tendon complex. Other elements of the model describe the body structure, or how it interacts with the environment. These may not always be considered part of the gait model but are vital for its implementation. These include joint limits and geometries, ground contact models, and constraints on degrees of freedom, and the location of muscle contact points.

Simulation

In this document, simulation is the process of using a computer to evaluate a model for a given set of inputs and initial conditions, over a period of time. The goal of a simulation is to reproduce both the internal and external behavior of the target system as closely as possible.

In this thesis, simulation describes a specific type of simulation, that is, forward dynamic simulation. This involves computing the dynamics of the body segments as they move through space. Time is divided into discrete steps, and at each time step, forces, accelerations, velocities, and positions are computed to determine the state for the next time step.

The Distinction Between Simulation and Model

We find it useful to clarify the difference between a simulation and a model. We will use simulation to refer to the execution of a program that computes the result of a model under specific initial conditions using a set of defined constants.

In contrast, the model is the abstract definition of the rules that the simulator follows. In academic papers, we represent models using equations, state diagrams, and block diagrams.

The simulation is the tool that a scientist uses to determine if the model is physiologically relevant. Comparing the simulations of two different models to data that is collected using motion capture or force measuring treadmills can help us determine which model more closely represents reality.

Optimization

The terms Artificial Intelligence and Machine Learning are common features of the public lexicon. In this context, when we describe model optimization, we mean to describe the process of using a machine learning algorithm to find a set of input parameters for our model that produce the desired result when simulated. In the context of gait, this generally means finding the set of tendon stiffness, controller gains, muscle force references, etc. that enable the simulation to emulate the desired gait.

Gait

A gait describes the manner of legged walking. Gait generally refers to the periodic leg movements found in locomotion. Examples of biped gait include walking, running, hopping. Quadruped gaits include trotting, galloping, cantering.

Challenges to Implementing a New Model from Scratch

The field of Biomechatronics lies at the intersection of several Engineering, Biology, and Medicine disciplines. The scope of a project can bridge several fields, and indeed, that is where the most interesting progress occurs. Forward dynamic modeling of gait is the place where high performance dynamics simulation and computing meet Anthropometry, Cellular Biology, and Neuroscience. As such, a researcher in with an idea for the latest and greatest gait model must master an array of skills before implementing a simulation to test their model.

Building a model simulation from scratch can be a herculean task, and it has been accomplished independently by the founders of this field and by recent researchers [8]–[11]. First, data must be gathered with motion capture and metabolism monitoring systems. Next, the researcher can begin to implement their idea for a gait model. But there is a catch. In order to simulate a lifelike gait model, one must make a simulation that simulates all the aspects involved. The simulation requires an implementation of a ground contact model to compute how the feet and floor interact. The simulation requires body segments and joints, whose positions and velocity can be set at initialization. The simulation requires an implementation of muscle and tendons that compute the force of a muscle based on its speed, length, and input stimulation signal.

After all of these things, controls architecture or other aspect of their model that makes it different from the ones that came before can be implemented. Maybe the ankle is controlled in a different way, or the tendon stiffness is computed by a new equation based on a new ultrasound study. This part of the work is the most critical, but it takes a small fraction of the time. After all, this novel part is the researcher's specialty, and they could probably explain this part of the model in their sleep.

The implementation does not end there. The simulation will need an assortment of analysis functions that can compare the kinematics to the motion capture data, and quantify the model success.

After all of the components of the model have been implemented and integrated, the researcher still needs to build machine learning code to search for the right set of input parameters to the model. Walking models are often so complex that they simply fall over without computer aided tuning of control variables.

At this point in the development cycle, the components of the simulator may have been tested individually, and they may compile as a group. But the critical assessment is producing a stable gait that can emulate the motion capture data. Any mistake in the above components can lead to failure at this system level, with little clue to what went wrong. Unfortunately, it may take days or weeks to find out if the optimization can find a working set of inputs.

So it is necessary not only to implement the new model, but also to implement a model with known behavior to test all of the components. If the well-known model does not work, then there must be a flaw in one or some of the components.

This type of all-or-nothing development is highly inefficient, and can lead to a debugging nightmare in the case of more complex gait models. It has been done before, and can be a valuable educational experience. In the context of simulating novel gait models, this development cycle is wasteful.

Challenges to Building Existing Implementation

At this point in time, some research groups have published source code to their models [9], [11], [12], and it is reasonable to build on top of their existing code. This methodology has its own problems. The specific model published may not be designed as the starting place for a more complex model. It is more likely to be a demonstration of their own model, than a general purpose platform for building different models. Also, the published model may not be able to perform a desired computational function, such as closed loop controls or parallel optimization.

The Goal

Our goal is to develop a toolbox for improving the efficiency of the model evaluation process, and thereby accelerating the development of future gait models. We aim to combine the

advantages of building with proven model components with the flexibility and specificity that comes from designing a model from scratch. We want to reduce development time by skipping the re-implementation of standard model components, and allow the researcher to spend more of their time evaluating and iterating on their chosen model component.

In academia, we see a high turnover of students, ideas, and code. But walking is a complex task, and simulating this task is complex. New students are faced with code that contains hundreds of constants and variables, often with context specific documentation. At this point in time, there are dozens of walking models in the literature, and these are often improved with small iterations on design and tests. But the real leaps occur when the best parts of different but successful models are combined. Therefore, it is useful to create a library of carefully categorized biomimetic components. These components can be tested in their original configuration, and then incorporated into other models. In this way, it is possible to rapidly incorporate the best elements from new models.

We aim to develop a framework for a library of model components and functioning simulators, so that a new researcher can focus on what matters.

Our system is a modular biomechanics physics engine capable of optimizing parameters in parallel on multiple cores. Each component of the system has a separate file and pre-defined interface. Thus it is possible to rapidly iterate on code changes, from things as specific as swapping out muscle models, to things as complex as changing the type of computer optimization used. Updates to the Simulink graphics utilities allow for immediate animation of the model.

Our one-feature-one-file approach has other benefits as well. First, it is conducive to collaborative efforts from multiple researchers, as each component of the neuromuscular simulation is described in a different file and can be developed in parallel. Second, it allows researchers to specialize their expertise on different component types, from ground contact models, to joint geometries, to neural controllers. Third, this system allows for time analysis of code. We are now able to generate reports of the runtime cost of different components in the dynamics model, and target the most computationally expensive components for revision. This leads to reduced computing costs, essential to staying at the cutting edge of computational modeling.

Methodology Overview

System Architecture

The methodology for designing the system is described in this chapter. This system is designed with the objective of improving the efficiency of the model evaluation process, and thereby accelerating the development of future gait models. First, the overall architecture is introduced. Later, individual components are described in detail.

The system has two major subroutines or subsystems. The optimization routine uses one of the Matlab machine optimization functions to search for an optimal set of input parameters for the model. The analysis routine is a short script that loads a results object and then calls the simulation and produces plots and an animation.

Simulink Concepts

To explain the concepts of this chapter, some Simulink features such as rapid accelerator versus normal simulation mode, and the mechanics of how Simulink works in those different modes must be understood. In short, when running in normal mode, Simulink compiles each time a simulation is run. It reads source files and writes executables to disk each and every time if parameters are being tuned. In rapid accelerator mode, Simulink creates an executable which can be fed certain tunable input parameters via an rtp structure. Currently, Simulink can not tune physical parameters related to SimMechanics blocks [13], [14].

Matlab can run some tasks in parallel. It does this by creating separate Matlab process 'workers' to complete these tasks in parallel. These workers are part of a cluster. A cluster can be set to run on one computer or across several connected computers, or on commercially available cloud

computers. In order to run Simulink simulations in parallel, careful consideration of Simulink file location and creation is necessary.

Optimization Subsystem

The optimization subsystem begins by calling an initialization script. During the initialization process, previous temp files are deleted. Next, the data necessary for the model is loaded, and the rapid accelerator executable and input data structure may be built if desired. Next, the optimization bounds are initialized, and an optimization function is called. The optimization function will repeatedly call the objective function, which will load model variables, change the tuned parameters, and then run the simulation. At the end of the optimization, the final output population and its corresponding heuristic scores are saved to disk in a small file.

The optimization subsystem can be run in four different configurations, resulting in different relative optimization runtimes. These configurations are listed in Table 1. All of the configurations listed are useful in certain situations; those omitted were not found to be useful. The green configurations are fast and useful for working model analysis. The red configurations are slower, and are more suitable for simulation development and debugging.

In Table 1, the runtime equations show big-O runtime analysis. In this table, g indicates the number of generations used in the machine learning algorithm, and n indicates the number of individuals in one generation of that algorithm. The multi-node cluster configurations show better runtimes because the number of cores in a cluster can be on the same order as the number of individuals in one generation. These equations assume that the number of cores on a desktop computer are not on the order of number of individuals in a population. The abbreviations R, P, M respectively stand for Rapid accelerator, Parallel, and Multi-node cluster.

Table 1: Optimization Configurations

Configuration	Rapid accelerator	parallel	Multi-node cluster	Optimization Runtime	Complexity
I - RPM	Yes	Yes	Yes	$O(g)$	Highest
II - PM	No	Yes	Yes	$O(g)$	
III - RP	Yes	Yes	No	$O(g*n)$	
IV - P	No	Yes	No	$O(g*n)$	
V - R	Yes	No	No	$O(g*n)$	
VI - null	No	No	No	$O(g*n)$	Lowest

Switching between Configurations

To switch into or out of rapid accelerator mode, one must alter which objective function is called. This can be done by modifying the referenced objective function in the project's machine learning wrapper function. Each project should have a rapid and a regular mode objective function. To switch into or out of parallel evaluation, simply toggle the relevant Boolean value in the same machine learning wrapper function. To disable the multi-node cluster, comment out the call to initialize_cluster in the initialize_all file.

Choosing a Configuration

During development, it is useful to start with the configuration of least complexity, and then increase to a faster configuration after successful testing of other configurations. It is useful to disable rapid accelerator and parallel in order to see warnings from Simulink and debug the objective function. It is necessary to disable rapid accelerator mode and use normal mode if the initial kinematics or aspects of anthropometry are tuned. Configurations that use large clusters of computers are more expensive to run, and are often difficult to impossible to debug if they err during optimization. These configurations should be reserved for validated Simulink models, as they significantly increase initialization time. It is important to note that the objective function should not output any warnings, errors, or print statements in order to run multi-node clusters as fast as possible. Print statements create a bottleneck at the master node, and slow hard drives can

create bottlenecks during normal mode. Shared cache or RAM can be limiting factors in parallel configurations if multiple cores being used.

Description of Configurations

Configuration I

The fastest possible way to run the optimization is in configuration I (Figure 1). In this configuration, the optimization subsystem begins by calling an initialization script. During the initialization process, previous temp files are deleted. Next, the data necessary for the model is loaded, and the rapid accelerator executable target and rapid accelerator target parameters structure, rtp, is built. The rtp structure and executable are saved to file. Next, the initialization script calls the project specific cluster initialization script. This initialize cluster script will start the cluster and move necessary files to worker nodes. These files include the rtp data structure, the rapid accelerator target executable, and any model specific function or script called by the objective function. Next, the optimization bounds are initialized, and an optimization function is called. The optimization function repeatedly calls the objective function in parallel, sending an array of parameters to each worker and receiving an array of heuristic scores. The objective function executes on a worker node. It loads the, rtp structure from disk, changes the tuned parameters, and runs the simulation in rapid accelerator mode, and computes the heuristic score. At the end of the optimization, the final output population and its corresponding heuristic scores are saved to disk in a small file.

Configuration I is most useful when the simulation takes at least several minutes to execute, and a large population size is needed for the optimization. In configuration I, a bottleneck can occur at the interface between the parent thread and the individual executions of the objective function. In Figure 1, the one-to-many process mappings are indicated with thicker lines. At the aforementioned interface, there is a one-to-many mapping, where one core is managing many.

To prevent this interface from capping the runtime, it is essential to eliminate print statements, and reduce the amount of data being passed during optimization. In this system, only the tuned parameters and the resulting heuristic are passed to and from the parent thread.

Figure 1 represents the fastest possible way to run an optimization. In this situation, a multi computer cluster is used by the optimization. Each processor core counts as a worker node, and separately computes the objective function. In this architecture there is minimal data being passed between the master and slave on objective function execution, because most of the data is passed during the initialization phase. This configuration is the fastest running version of our design, and is ideal when debugging is complete and a large generation size is needed.

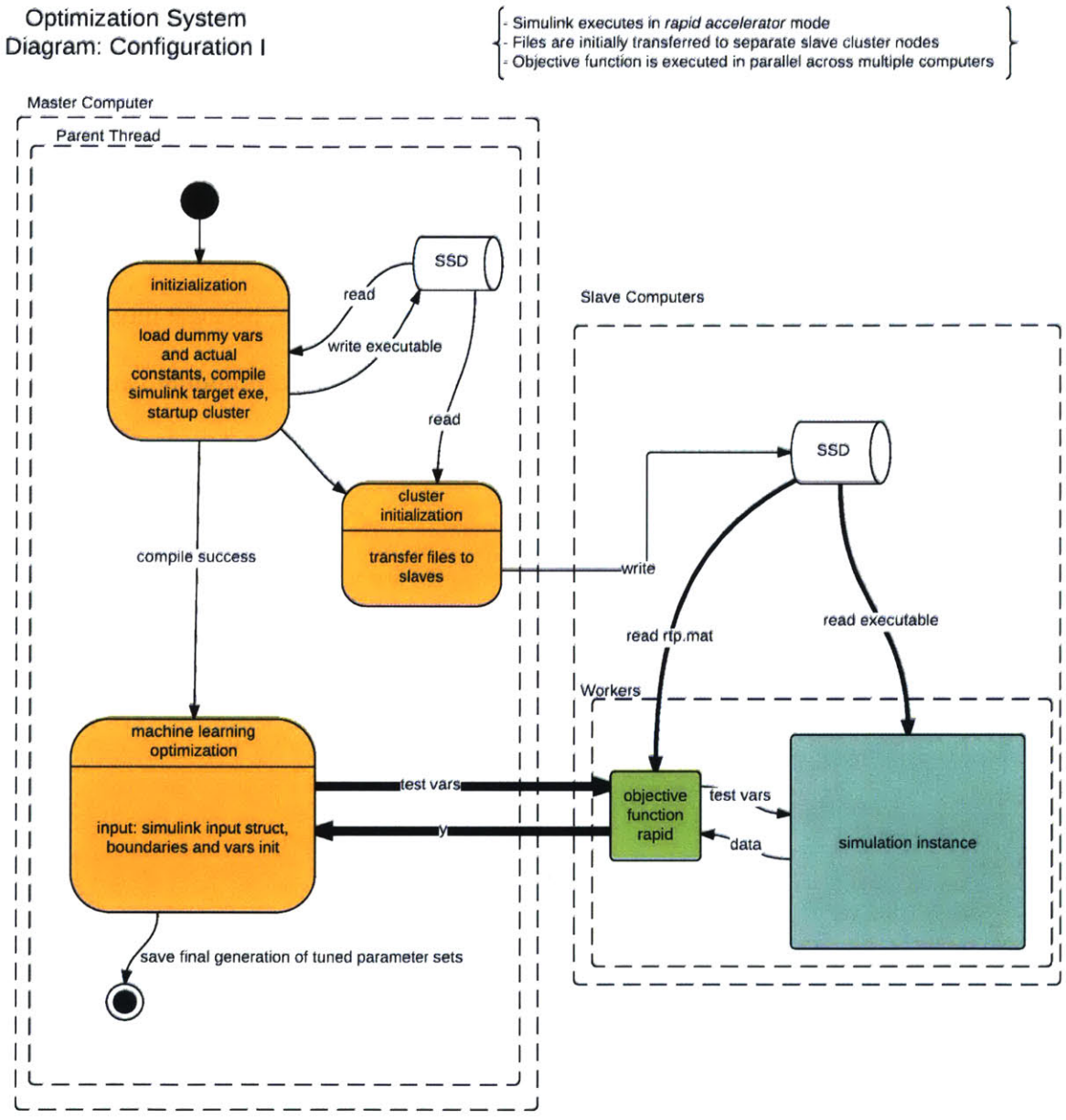


Figure 1: Configuration I

Configuration II:

If the initial kinematics or aspects of anthropometry must be tuned, configuration II is the next-fastest option. Simulink is not able to make these parameters tunable in the rapid accelerator target, so it is necessary to run the simulation in normal mode.

The optimization subsystem begins by calling an initialization script. During the initialization process, previous temp files are deleted. Next, the data necessary for the model is loaded, and all files used during the objective function are transferred to each worker computer in the cluster. Next, the optimization bounds are initialized, and an optimization function is called. The optimization function repeatedly calls the objective function in parallel.

The objective function loads the constants for the simulation, changes the tuned parameters, runs the simulation in normal mode, and computes the heuristic score. At the end of the optimization, the final output population and its corresponding heuristic scores are saved to disk in a small file. This process is described in Figure 2.

In normal mode, the objective function is slower because Simulink re-compiles the model each time the simulation is run. This explains the increase in calls to and from the worker hard drives as shown in Figure 2. With normal mode, Simulink will write to the hard drive during the execution of the objective function. Thus it is useful to have a fast write speed in this situation, and to pay attention to the number of workers per computer.

This figure illustrates the points at risk for becoming a bottleneck with the line thickness increase. This configuration is the fastest way to optimize model parameters if the initial mechanical conditions are being tuned.

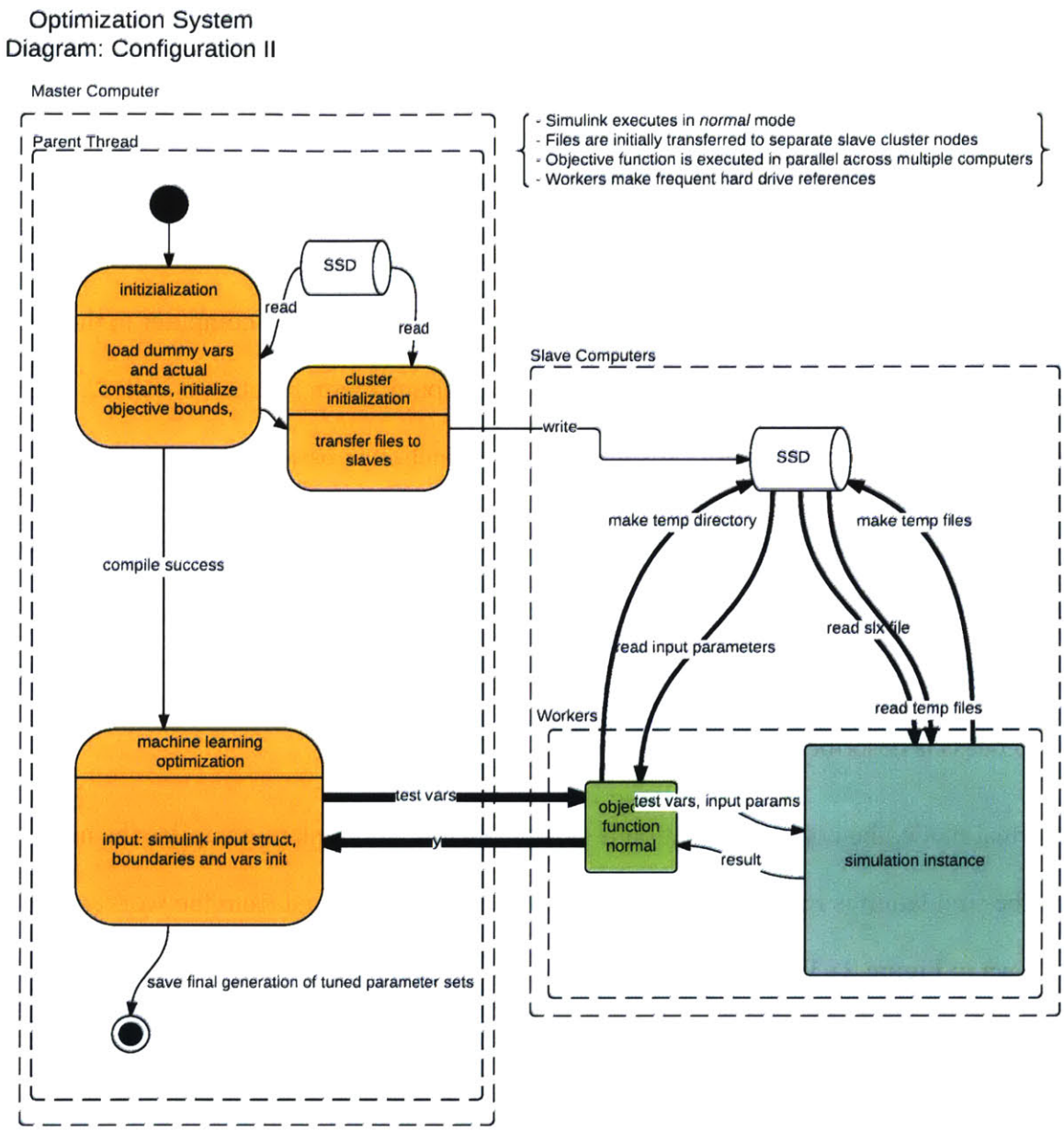


Figure 2: Configuration II

Configuration III:

In this configuration, the optimization subsystem begins by calling an initialization script. During the initialization process, previous temp files are deleted. Next, the data necessary for the model is loaded, and the rapid accelerator executable target and rapid accelerator target parameters structure, rtp, is built. The rtp structure and executable are saved to file. Next, the optimization bounds are initialized, and an optimization function is called. The optimization function repeatedly calls the objective function. The objective functions will execute on their own cores in parallel. The objective function loads the, rtp structure from disk, changes the tuned parameters runs the simulation in rapid accelerator mode, and computes the heuristic score. At the end of the optimization, the final output population and its corresponding heuristic scores are saved to disk in a small file. This process is described in Figure 3.

With the right computer setup, Configuration III may execute fast enough to be the preferred configuration for a developer. It may be useful to set up a computer with a fast read speed disk, and a large cache instead of paying for cloud space or maintaining additional machines for configurations I and II.

This configuration may be preferred for its combination of simplicity and its quick execution time on desktop machines.

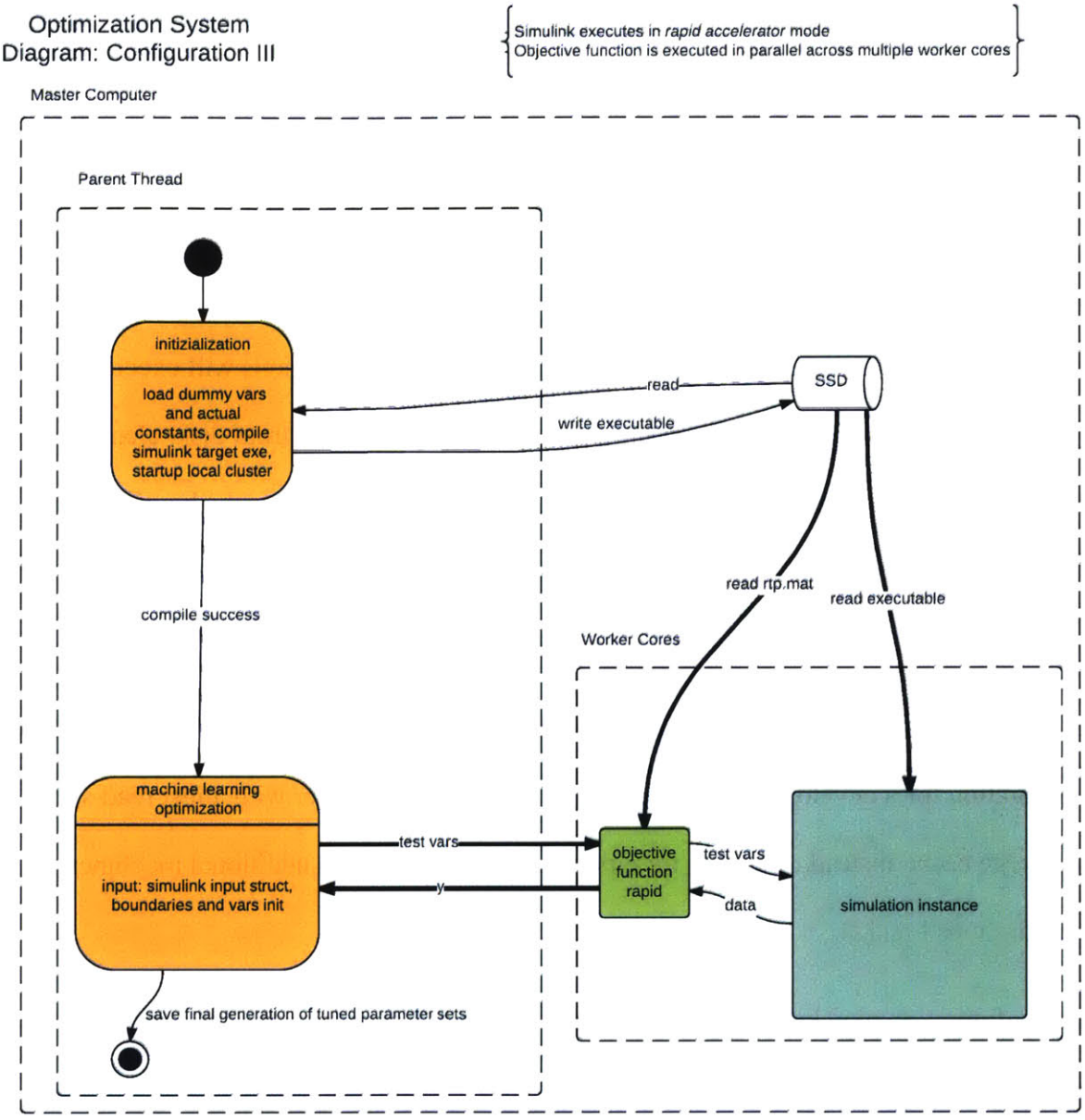


Figure 3: Configuration III

Configuration IV:

Configuration IV is similar to Configuration III, with the exception that Simulink runs in normal mode for Configuration IV. Simulink is not able to make certain mechanics parameters tunable in the rapid accelerator target, so it is sometimes necessary to run the simulation in normal mode. If the initial kinematics or aspects of anthropometry must be tuned, configuration IV is the next-fastest option.

The optimization subsystem begins by calling an initialization script. During the initialization process, previous temp files are deleted. Next, the data necessary for the model is loaded. The optimization bounds are initialized, and an optimization function is called. The optimization function repeatedly calls the objective function in parallel.

The objective function loads the constants for the simulation, changes the tuned parameters, runs the simulation in normal mode, and computes the heuristic score. At the end of the optimization, the final output population and its corresponding heuristic scores are saved to disk in a small file. This process is described in Figure 4.

In normal mode, the objective function is slower because Simulink re-compiles the model each time the simulation is run. During the compile process, Simulink writes multiple files. If there are multiple workers running in parallel, they must write those files to a temp directory to avoid collision. This explains the increase in calls to and from the worker hard drives as shown in Figure 4. With normal mode, Simulink will write to the hard drive during the execution of the objective function. Thus it is useful to have both a fast write speed, and a fast read speed in this situation, and to pay attention to the number of workers per computer.

Once again, the diagram illustrates the points at risk for becoming a bottleneck with the line thickness increase. Thicker lines indicate a bottleneck where many processes communicate with one single process or module.

This configuration may be preferred for its combination of simplicity and its quick execution time on a single desktop machine. It must be used when the initial mechanical parameters of the simulation are being optimized.

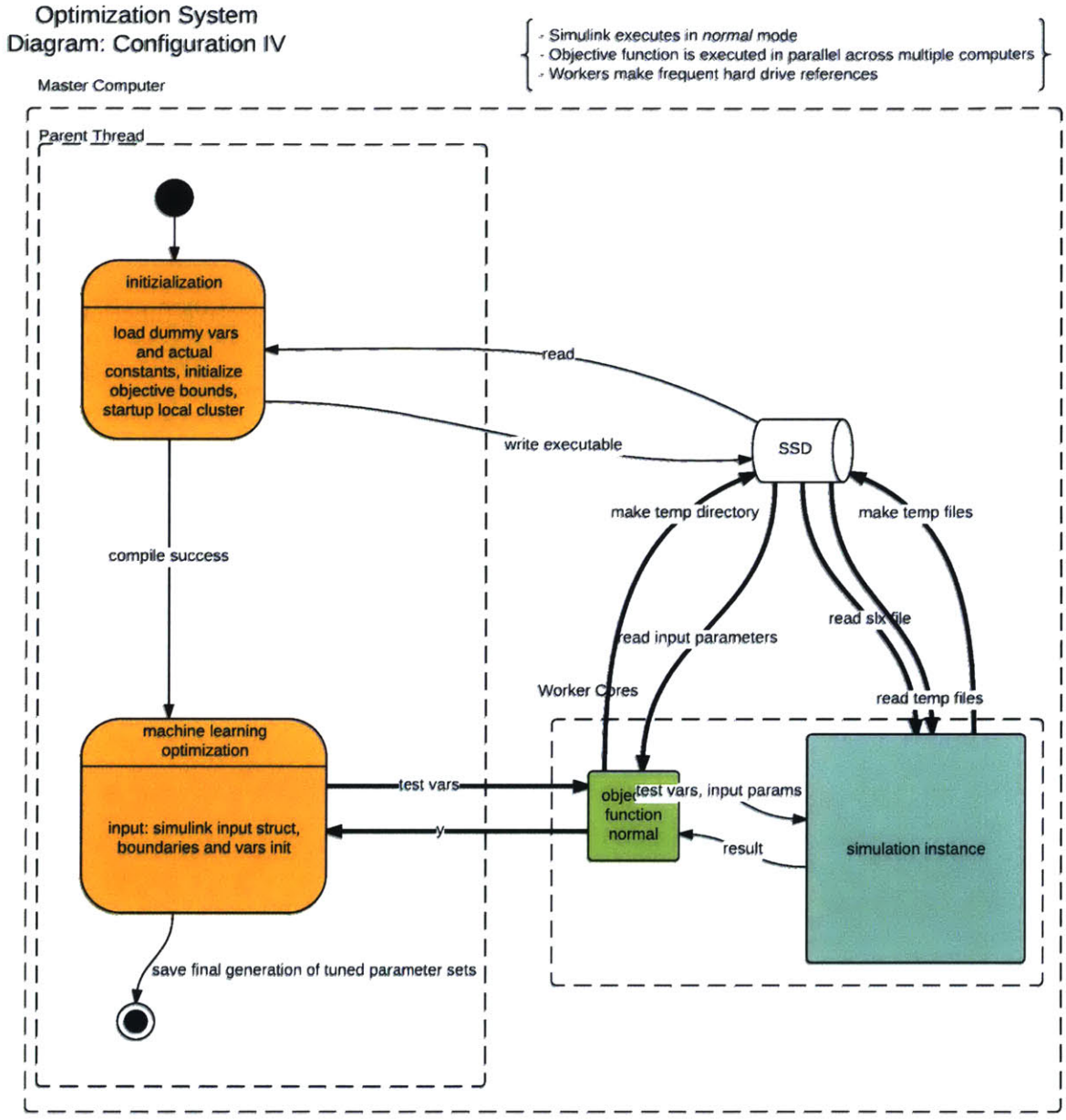


Figure 4: Configuration IV

Configuration V:

Configuration V is a serial execution, rapid accelerator configuration. Configuration V is useful for running an optimization on a single core and testing the rapid accelerator version of the objective function. It is also useful for running the optimization on a mobile device, such as a laptop. This configuration is a good debugging configuration for ensuring that the rapid mode works well in optimization, and that there are no problems with the objective function.

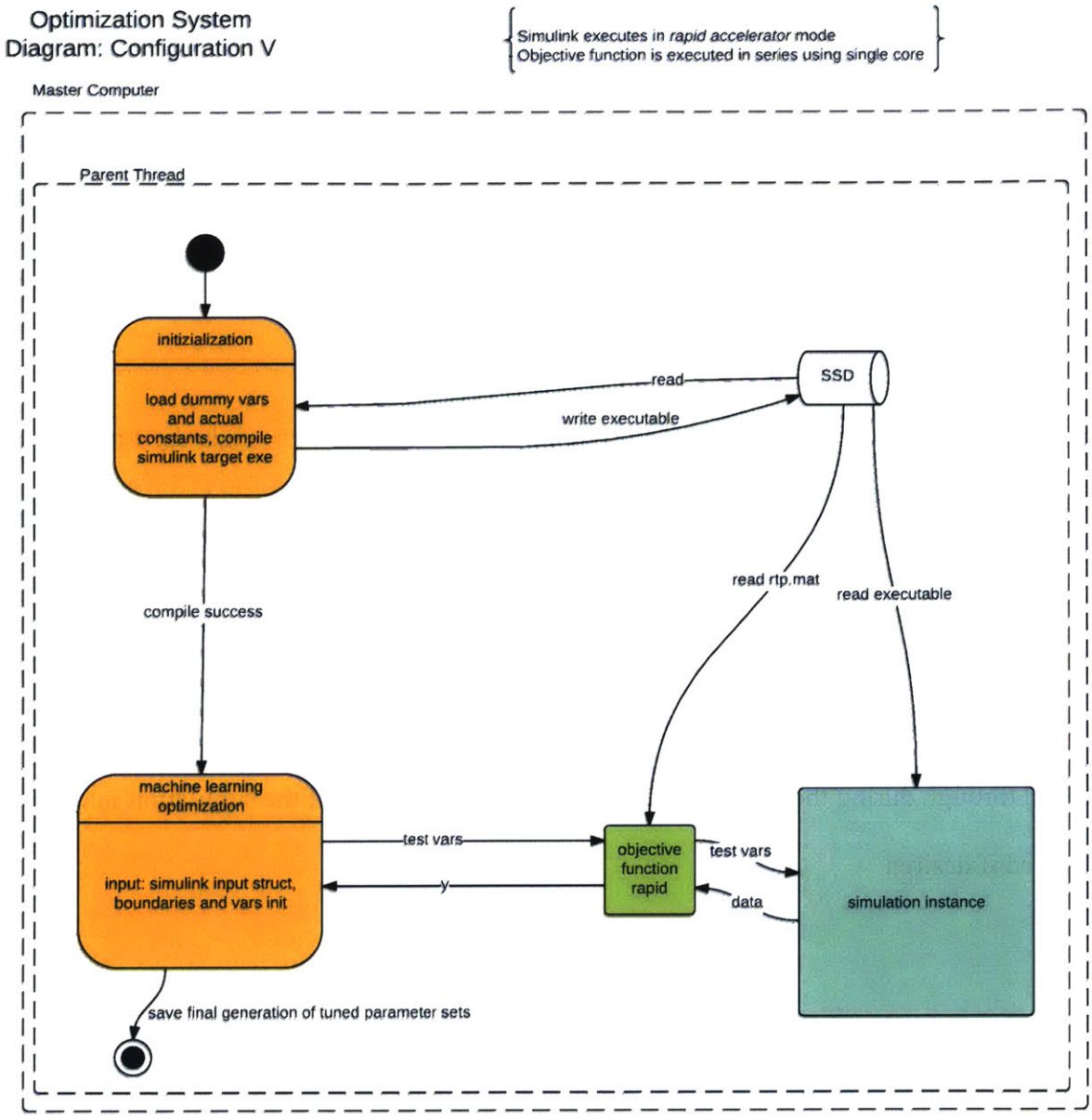


Figure 5: Configuration V

Configuration VI:

Configuration VI: This configuration is the slowest possible configuration. It runs the optimization by evaluating the objective function in series. This mode is useful for debugging the simulation itself, when there is a bug that is can only be reached part way through the optimization. This mode is also useful for testing and developing the normal mode objective function.

For testing purposes, it is often useful to set the genetic algorithm to a single digit number of generations and population size. This way you can run through the entire program and discover any problems that exist at the system level without waiting for a real optimization to complete.

This configuration is a good debugging configuration, but it is very slow. This configuration is useful to debug the simulation. Warnings from Simulink are visible, the execution can be stepped through during the objective function, and each iteration of the simulation may be animated if desired.

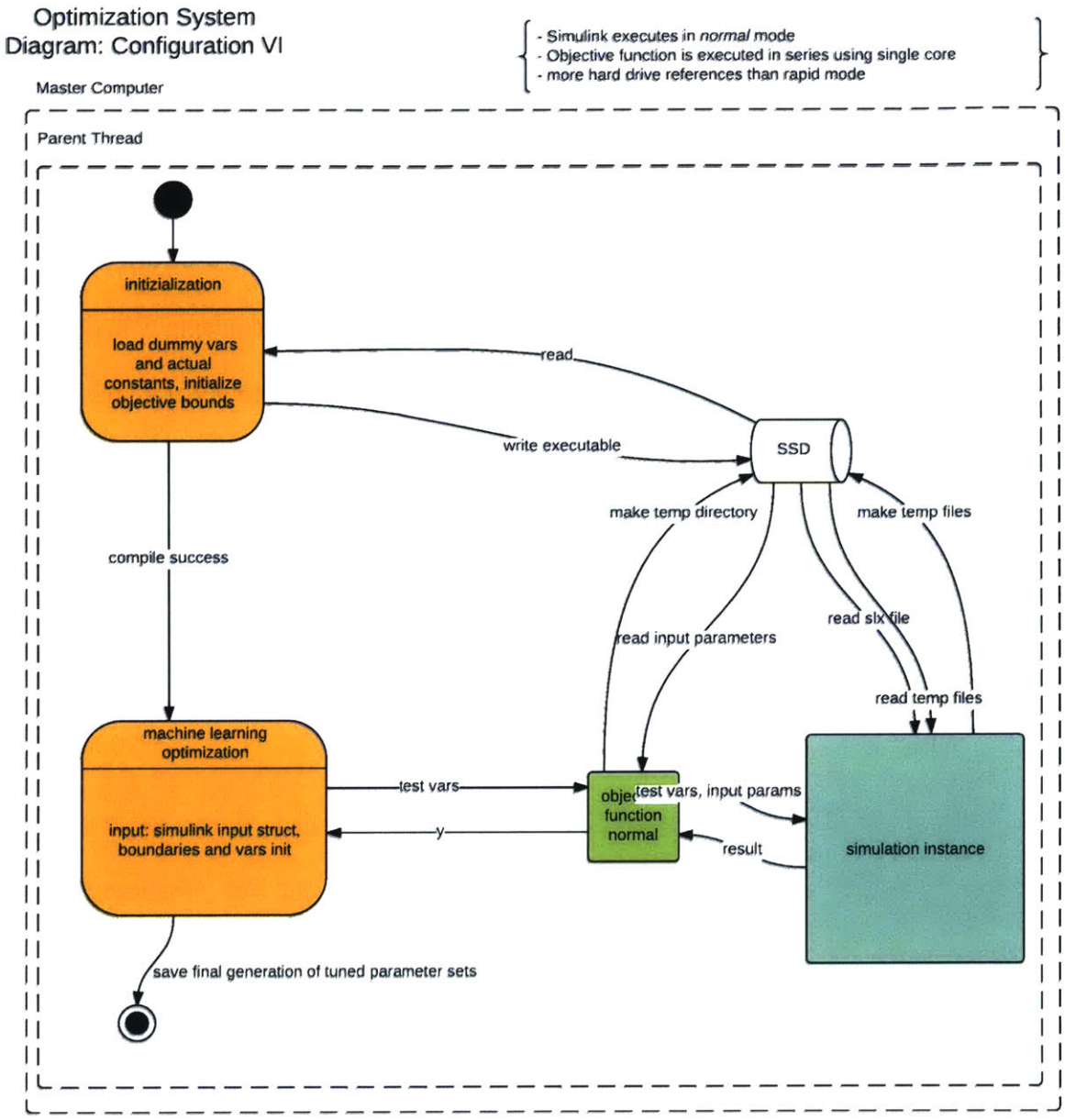


Figure 6: Configuration VI

Analysis Subsystem:

The Analysis subsystem is designed to run after an optimization. It loads the saved file which contains the population and the scores of each individual. The analysis script then selects some individual from that population, and runs a simulation in normal mode, and then performs

analysis on the results. During the final stage of a project, it is useful to fork the Simulink file to produce one outputs more data. A version such as this would be impractical for use during the optimization, but can be implemented once the Simulink model has reached sufficient design maturity such that it is not frequently changed at the top level.

Figure 7 explains the architecture of the analysis system. This system is used after the primary optimization is completed. The key to this system is its simplicity and adaptability. The analysis script follows a similar template for each model, but can be modified to generate desired plots. The rendered video is generated by the Simulink instance, thus minimizing the development time spent on animation.

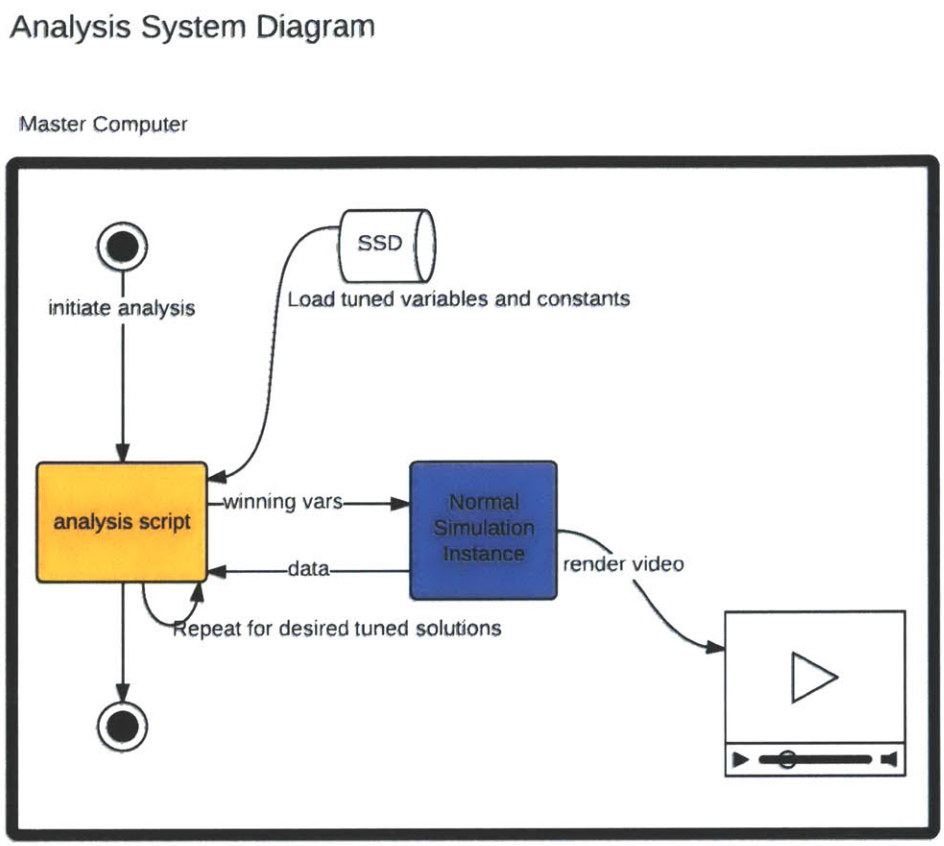


Figure 7: Analysis System Diagram

Multi-Project Organization

Because this system must accommodate multiple models being researched simultaneously by multiple individuals, file structure and version management are critical aspects of the design. A file structure is used where multiple developers can work on different projects in parallel, setting the Matlab path to suit their needs, but working from the same root directory and the same version controlled codebase.

The root-level folder contains template 'm' files for new projects, the library folder, and each project folder.

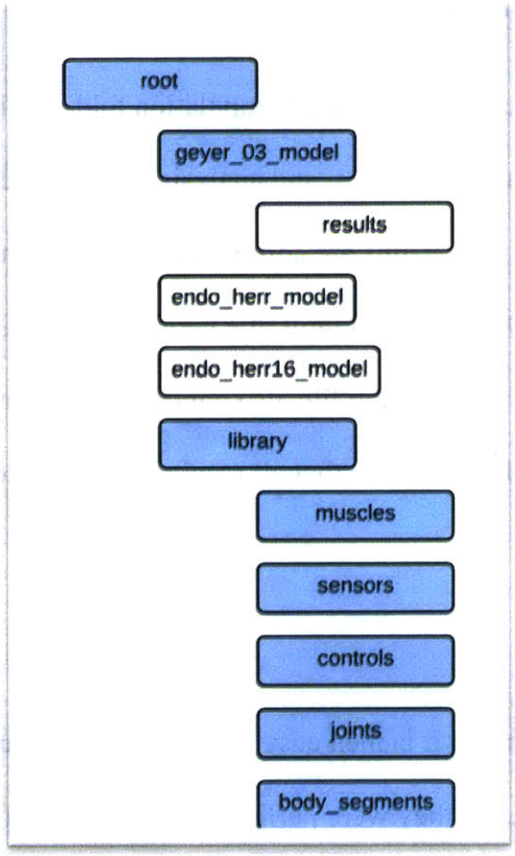
A project folder contains all non-library components needed to run the model. This includes modified template '.m' files, the Simulink model, a results folder, and prototype library components.

The root-level library folder contains tested model components such as muscles, sensors, controls, joints, and body segments. Each time a project implements a new component, it is added to the shared library after sufficient validation.

Organization File Structure

Figure 8 illustrates the file structure used to maintain project separation while maintaining common library use and inter-project collaboration.

Figure 8: Organization File Structure



Path Management

This method of file structure leads to the potential for file name collisions. Individual project folders each contain files with the same name that other project files have. For example, each project should have a model, objective function, main function, etc., and each of those files will have the same name. As such, it is necessary to carefully manage the Matlab path in order to exclude same-name files that exist in other projects. An example file structure is shown in Figure 8 with the path highlighted.

Results Organization

Keeping track of optimization results can be a challenging aspect of this type of work. The population and scores are only meaningful in the context of the exact version of the simulation used to find them. The results are also the most valuable product, and thus it is useful to pay consideration to their management.

It us useful to put a results subfolder into each population folder. Then, each time an optimization is run, the population, scores, and analysis plots can be stored in a subfolder of the results. At this point, the project results are committed to version management, and the entire project is also committed. During the commit, the version number is noted, and recorded in a readme file with the results. The readme file also contains any interesting notes about the optimization. This methodology ensures that all results are reproducible, and that extra commits are made at useful times.

Revisiting Results

With proper results management, it is then easy to perform results analysis on any data set, and revisit old optimizations, even if they were performed. Simply navigate to a new directory and check out the relevant version that is noted in the results folder.

This capability is key for rapid development in projects. Now, it is easy to change the model and make progress, without concern for losing the ability to perform more analysis on an old data set, or having that parameter set produce different results in a modified model.

Typical Workflow

The typical workflow for developing a new model or model improvement is steered by the system architecture. First, the researcher, let's call her Alice in this hypothetical example, conducts a review of the models that are similar to the one she wants to implement. Alice identifies similar models and the important differences. Next, Alice checks out the latest version of the simulator files onto her computer. She now has access to all of the projects being developed by the lab and to older projects that have published papers describing them. For this example, Alice has chosen to improve a model designed by Bob by changing the controller. Lucky for Alice, Bob has used this system to implement his prior model.

When she is ready to begin the implementation, Alice makes her own working project directory in the root folder, complete with a readme file describing the project and identifying herself as the developer. Next, Alice navigates to the prior project, chooses a results file to copy, and copies the working files from the version referenced by the results. Alice now has a valid and tuned model to start with, and can make changes so it matches her goal model.

In this example, Alice wants to change the controller, so she creates a new Simulink library file and creates a masked block with her controller logic inside, a dialog box for input parameters, and input and output signal ports. Next, Alice adds that library based controller to her model in place of the old controller. She adds new parameters to the appropriate lists, and she runs the existing analysis script with some test values as inputs to the controller.

Next, Alice chooses which model parameters will be based on research or textbook values, which parameters will be based on prior optimization results, and which parameters will be

optimized. She updates the lists of optimized parameters and then runs the optimization. In configuration VI with only two generations and two individuals.

Once the optimization is complete, Alice can finally perform analysis on her controller, and compare it to prior results. Since the only thing to change between models is the controller in question, Alice can be confident that her changes are what precipitated any changes in the result.

Once Alice has validated her new control, she can move her library file from her project folder to the shared library.

Specific Technology Challenges

In the implemented force feedback hopping project, initial kinematic state and muscle attachment locations were the only parameters that were not possible to tune in rapid accelerator mode.

In some projects, it is not necessary to tune initial kinematic state because it is preferable to take the initial conditions from motion capture data.

The muscle attachment location is more important to tune because model performance can be heavily skewed by suboptimal placement. This is an interesting situation because, of course, biological tendons do not attach to infinitely small points.

The inability to efficiently tune mechanics parameters in Simulink's rapid accelerator mode is a big drawback. It is possible to design a next-generation library where the muscle component simply connects to the center of mass (COM) of the rigid body and applies both a torque and a force. In this situation, the muscle block would take tunable parameters for the position of the

attachment points on each block relative to the center of mass. The muscle would measure the distance between the two bodies and their orientations to compute the muscle dynamics.

Building a Library: Simulink Model Elements

Introduction

This chapter explains the library implementation of the different model components. It was not possible to describe all of the components individually in this document. A representative subset was chosen.

Simulink is a powerful and flexible modeling tool. As a result, there are sometimes many ways of implementing a model, and the easiest way is not always the best or most efficient way once all of the elements of the research cycle are considered. For this project, a review of best practices in Simulink and Matlab was conducted in order to establish guiding principles for component design [15], [13], [16], [14], [17].

Building a Model: Root Level Canvas Examples

Using this framework, building a model is a process of implementing library components and assembling them together on to the root level canvas. The root level canvas is the most complex space in the system, with many links between blocks, and many parameters in the same space. Figure 10 illustrates the level of complexity reached with a full walking model. With a deliberate naming system for the variables and tidy block placement, even systems with many communicating components can be built.

With a system as interconnected as in Figure 10, it is tempting to resort to using tags instead of lines to make the signals look neat. We have found that this technique obscures the naturally complex relationships between components, and makes code maintenance more difficult. The problem with tags is that they add an extra set of names to the system, and it is time consuming for those unfamiliar with the code to trace tags.

By avoiding tags and by masking subsystems, there are only two ways to pass variables into and between subsystems in Simulink: parameters and signal lines. The signal lines can be seen, and are obvious. Passing parameters in Simulink is similar to passing parameters in other programming languages. A masked subsystem only has access to the parameters specified in its mask. To pass a parameter into a subsystem (e.g. Figure 12), double click the block, and the programmed mask dialog box will pop up (e.g Figure 13). At first, a mask dialog box is populated with default parameters (e.g. Figure 16). This can be changed to the appropriate variable name that will be in the accessible workspace (e.g. Figure 13).

Parameters are used to transfer variables from a parent workspace to a child workspace. For example, when the model is called from Matlab script, it is set to use the 'current workspace.' This means that the variables in the calling Matlab function are accessible by the root level of the model. These variables can be passed into a masked block via the mask dialog box, where they can also be renamed.

When subsystems are masked and saved as library blocks, they can be shared by multiple models. For example, the Endo-Herr model and the force feedback hopping model share ground contact models, sensors, body segments, muscle elements.

Resolution Disclaimer for the Following Figures:

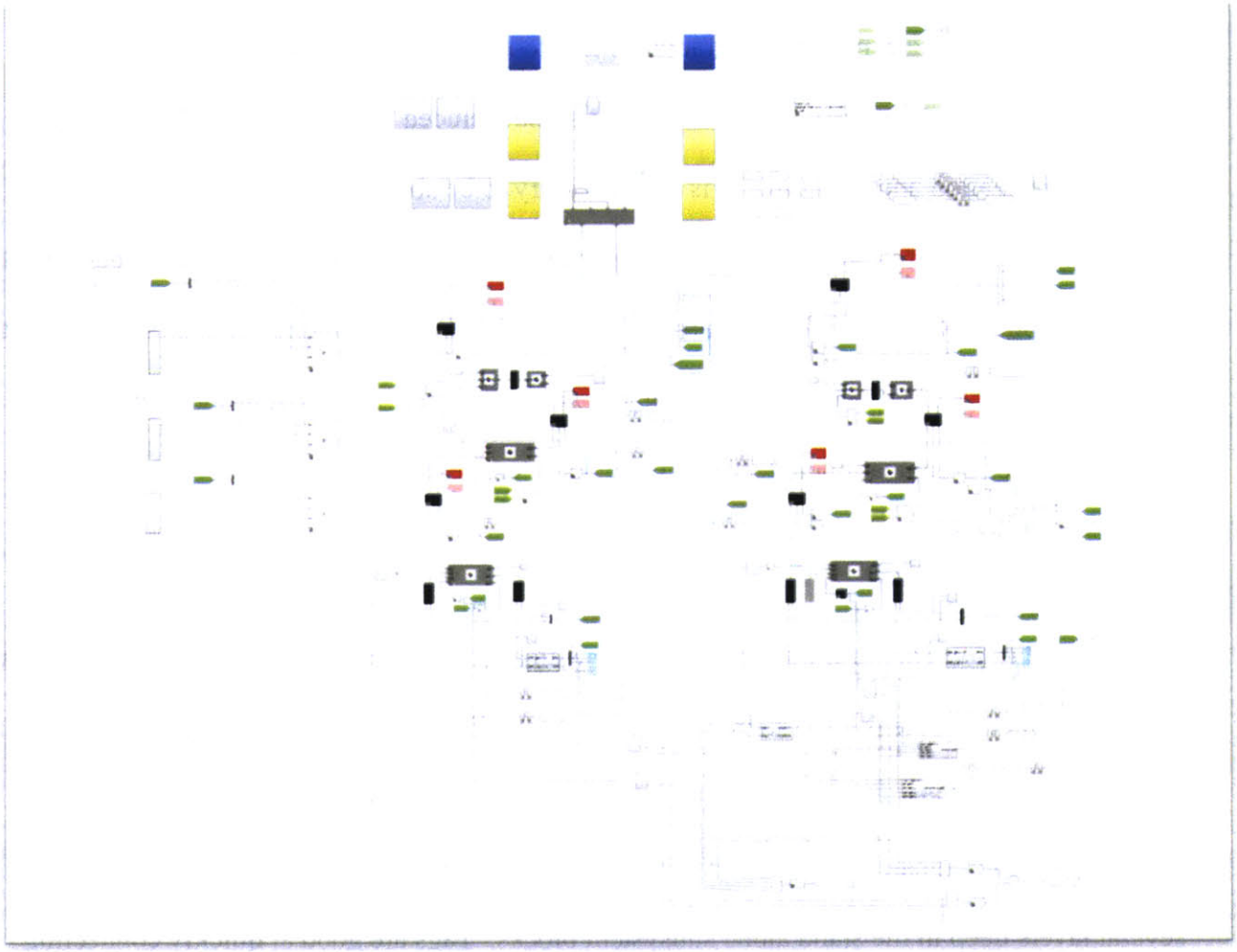
Some of the following Simulink Diagrams contain too many components to be reasonably displayed in single page form. In these situations, the included figures serve as illustrations of general organization and structure. It is not necessary to be able to see every label in the diagrams. These figures are included for the purpose of conveying the general techniques used in this methodology, and not for the purpose of transmitting code content. It is okay if it is not possible to read all of the words in the Simulink canvas images. For most prupose, only the general sense of organization and structure are being conveyed.

Before Modular Design:

As discussed in the Forward, this project began with another code base. In order to illustrate the improvements upon the prior system, a diagram of my old root level canvas is included in Figure 9 for comparison. One who is familiar with Simulink tags may notice their shape in green. These tags communicate not only with other element on this canvas, but also with components several layers deep in the stand alone subsystem blocks. Tags are good at quickly connecting parts code, but they make it difficult to trace relationships later on.

This implementation is layered, with standalone subsystems communicating via tags. Much of what is shown here is code used for analysis of the model and for animation in external code. This root-level snapshot is just the tip of an iceberg of interconnected subsystems.

Figure 9: Before Modular Design



Modular Design Implementations:

Figure 10 illustrates the root level canvas for a full walking model [18]–[21], which is currently under development. This canvas contains the most complex layer of the model where all of the subsystems are linked together, and where IO to Matlab is established. This diagram can be used as a direct comparison to the diagram Figure 9. While the previous root level only showed the tip of the iceberg, with Tags communicating to unseen subcomponents, this new methodology shows all of the complexity in one place. Using this method, the code can be used to communicate the structure of the model.

Figure 10: Endo-Herr Root Level Canvas

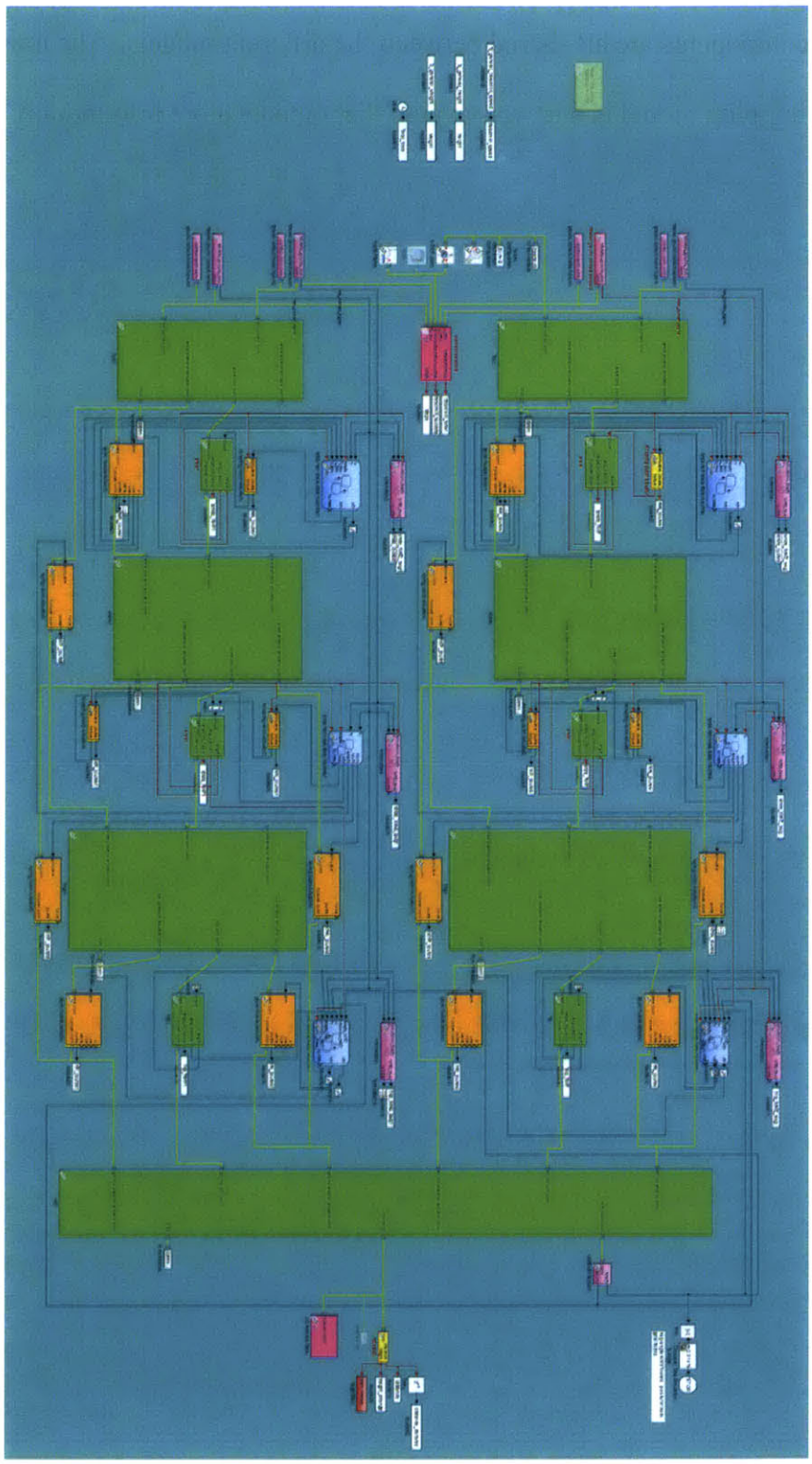
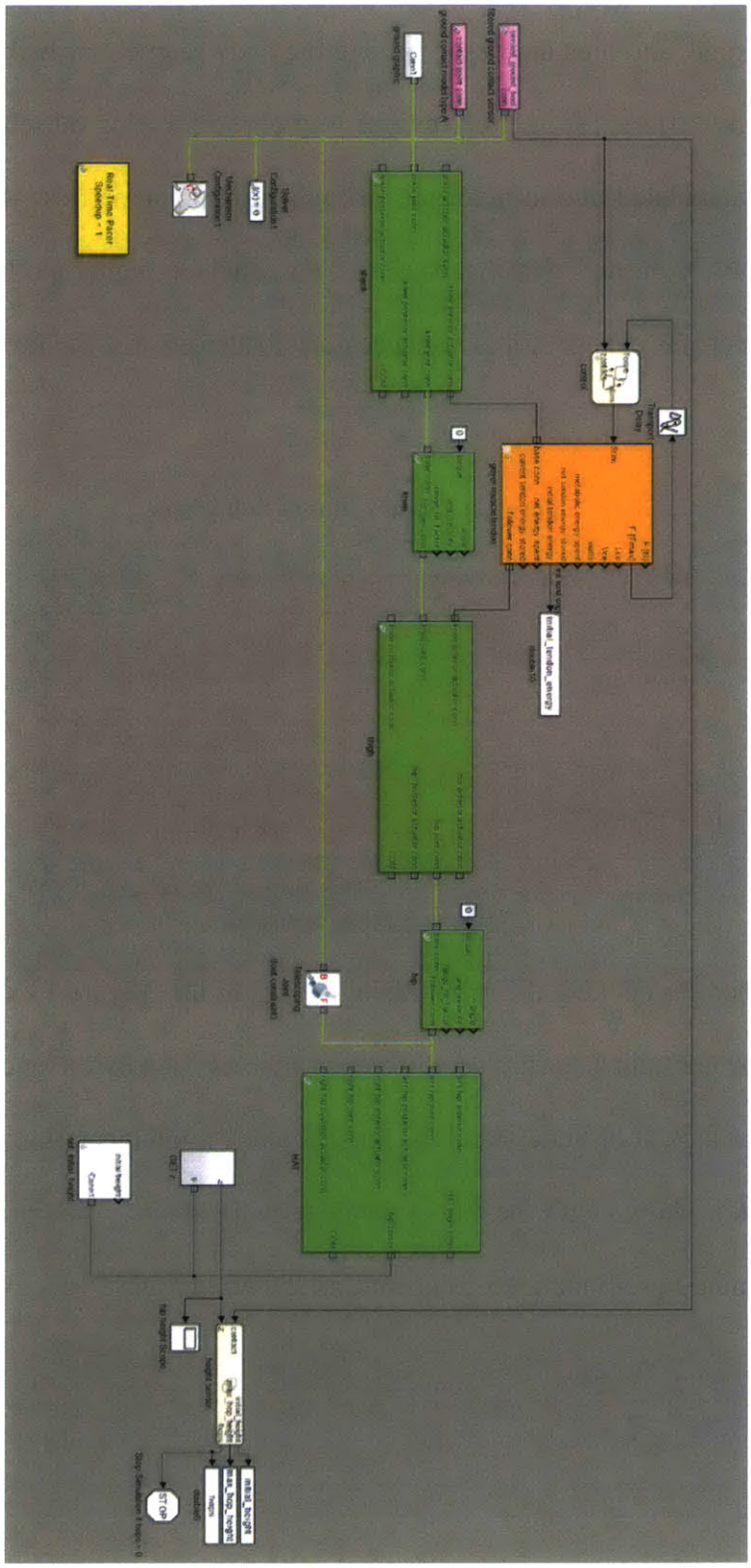


Figure 11 depicts the root level canvas for the root level hopping model. This model is used to produce the results in this thesis. This implementation contains two leg segments, connected to a

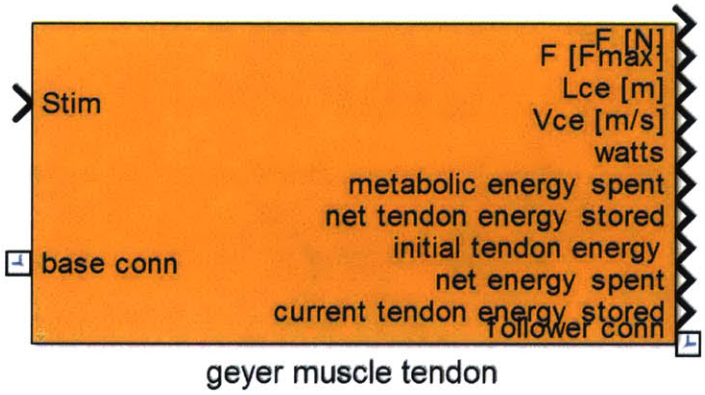
HAT segment. The Joint blocks, thigh and HAT body segments, ground contact and ground contact library components are all shared between the different models. The muscle used in the force feedback hopping model is a newer version that outputs more information.

Figure 11: Force Feedback Hopping Canvas



The muscle block shown in Figure 11 is shown below in Figure 12. This muscle block takes an input stimulation signal, and then applies a force to the body segments which it is connected to. This is fundamentally different from the previous way of representing muscles, in that the muscle force is not translated into joint torque through custom geometries. Instead, the muscle acts directly on the body through SimMechanics. This results in higher modularity. When the body segment geometry is changed, it is not necessary to also change the lever arm length in the muscle code.

Figure 12: Library Muscle Subsystem



When the muscle block is clicked, the mask dialog box pops up. Figure 13 shows the dialog box. The dialog box contains boxes to set different variables used by the underlying subsystem. It is not necessary to look at or understand the muscle block implementation to understand how to use it. This block is shown with the model parameters filled out. The library components initially contain standard pre-populated constants, as shown in Figure 16.

Figure 13: Muscle Mask Dialog Box

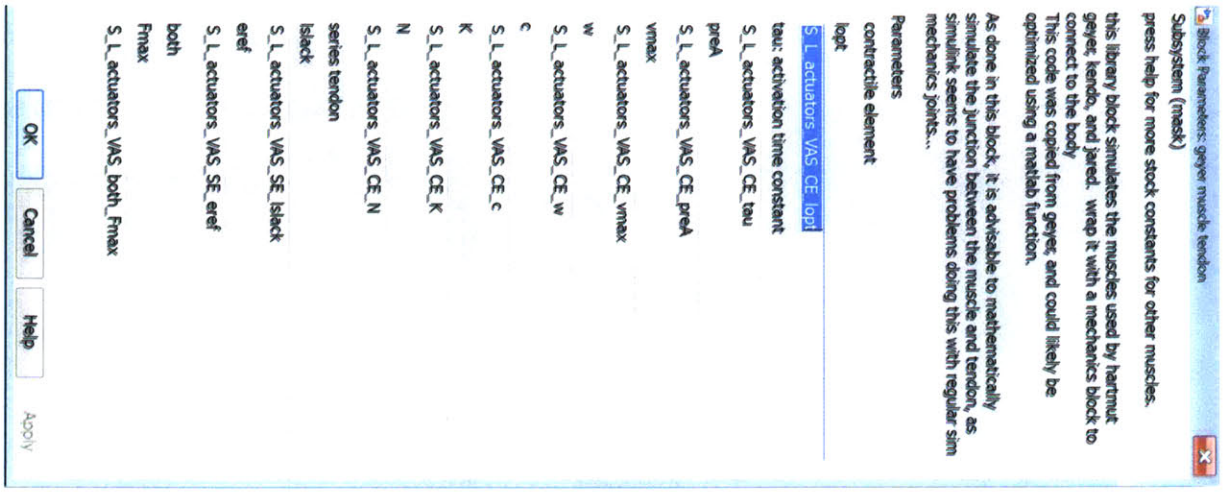


Figure 14 depicts the implementation of the library muscle. This subsystem contains scopes for debugging, and inner subsystems for the actual force and metabolism calculations. This canvas shows how the muscle applies the forces to the body segments in the SimMechanics 2.0 modularized paradigm.

Figure 14: Library Muscle Implementation

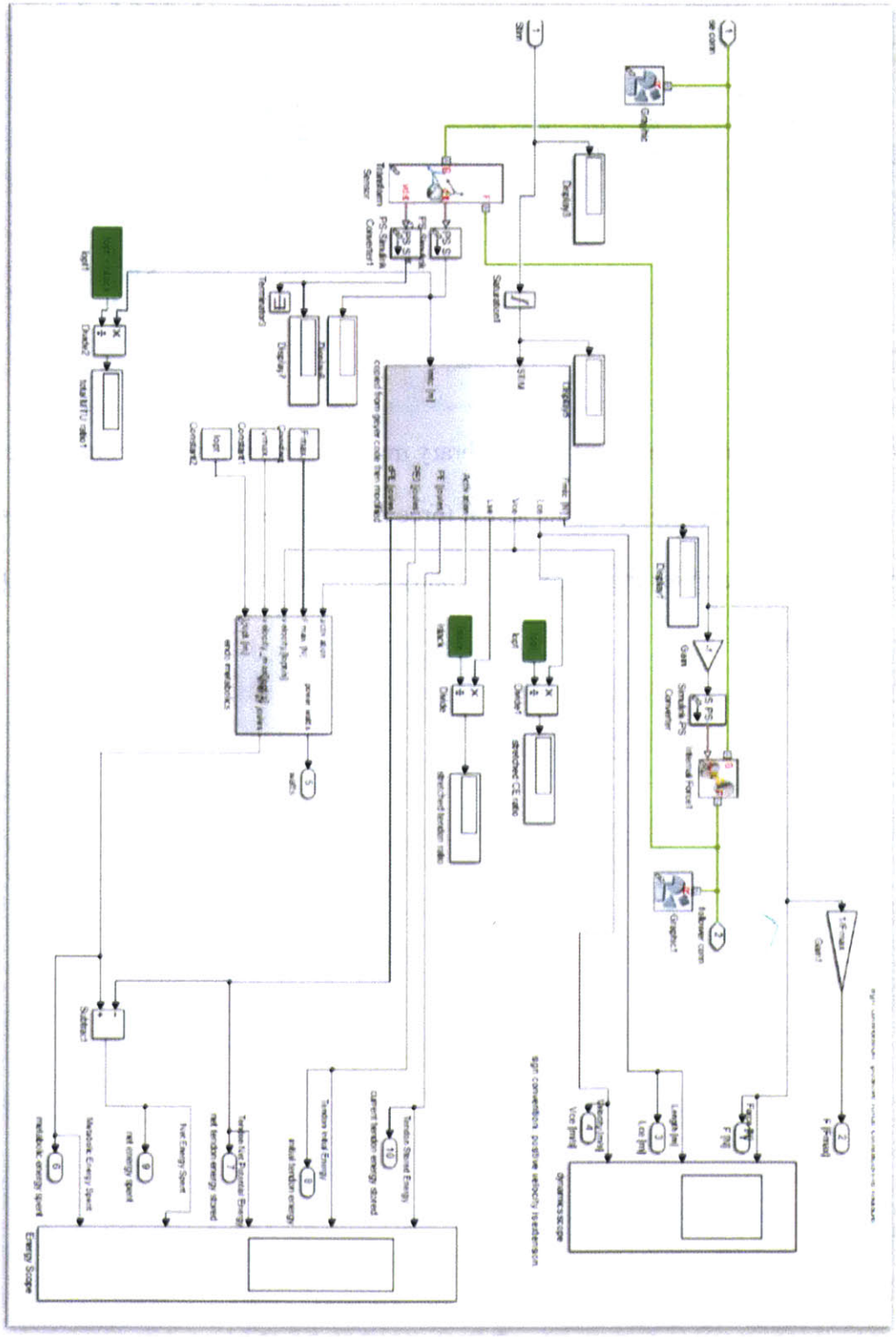


Figure 15 shows the sagittal plane thigh library component. This component is based off of textbook constant scaling [22]. The block has connection points for muscles and joints, and a connection point to the COM for analysis or more abstract calculation-based force and torque manipulations.

Figure 15: Sagittal Plane Thigh Subsystem

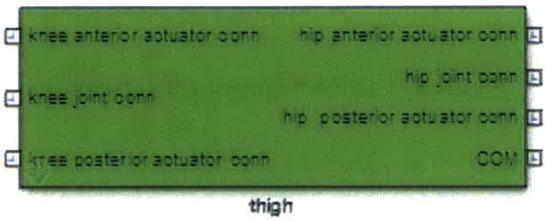
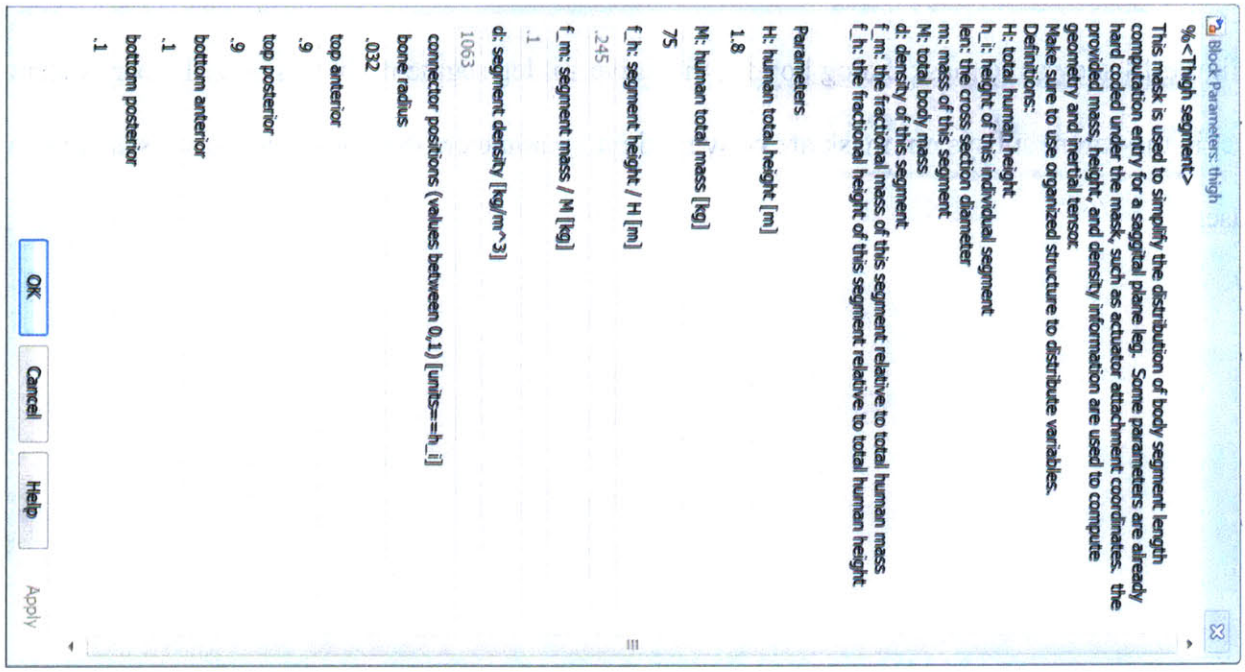


Figure 16 shows the mask Dialog box for the thigh subsystem. Each box comes pre-populated with a reasonable value, based off of textbook sources [22], [23]. Tendon attachment locations are based on measured values from the photographic atlas images.

Figure 16: Thigh Mask Parameter Dialog Box



The thigh segment is one area where there are benefits of nested library components. In terms of implementation, the thigh and shank are very similar. Both have rectangular shape and parameters. The implementation of the thigh was thus a matter of putting an extra mask over a universal leg segment. The canvas of the thigh is shown in Figure 17. Most of the work here is cone in the mask implementation, where constant parameters such as the fractional height of the thigh segment relative to the body are set.

Figure 17: Thigh Segment Implementation

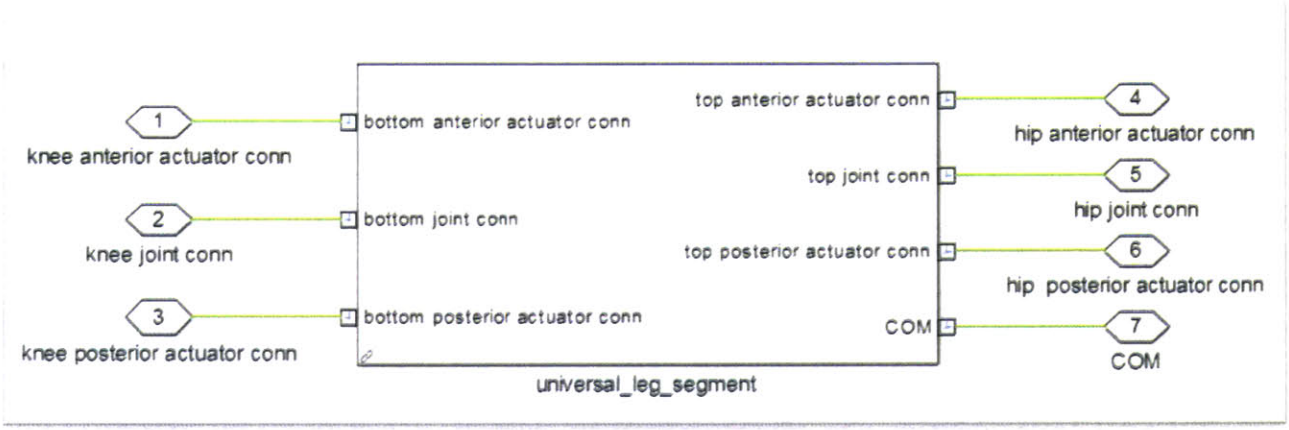


Figure 18 shows the mask dialog box for the universal leg segment. In this box, the parameters set in the parent subsystem mask are converted into parameters that the enclosed subsystem can use.

Figure 18: Universal Leg Segment Mask Dialog Box

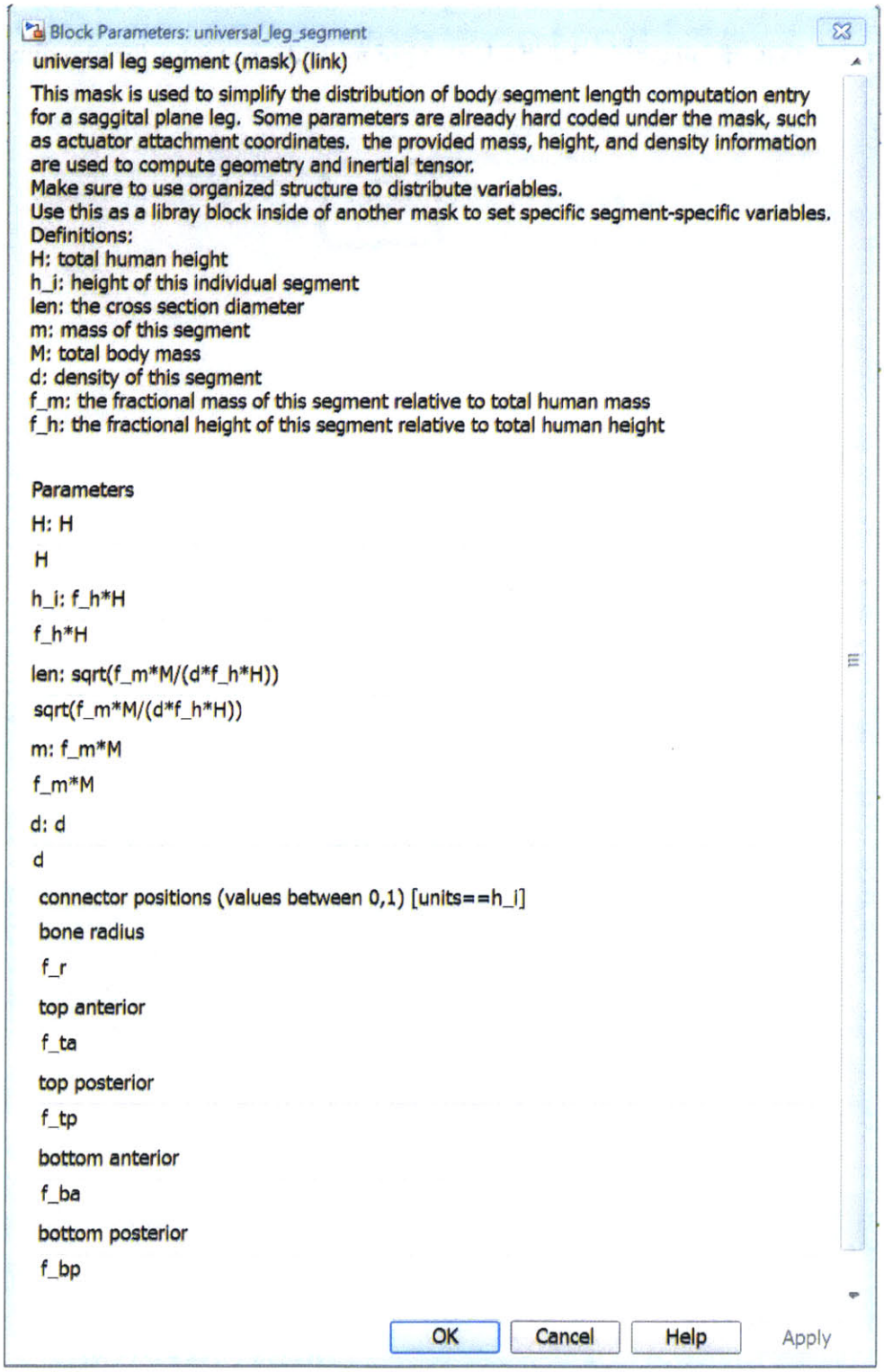
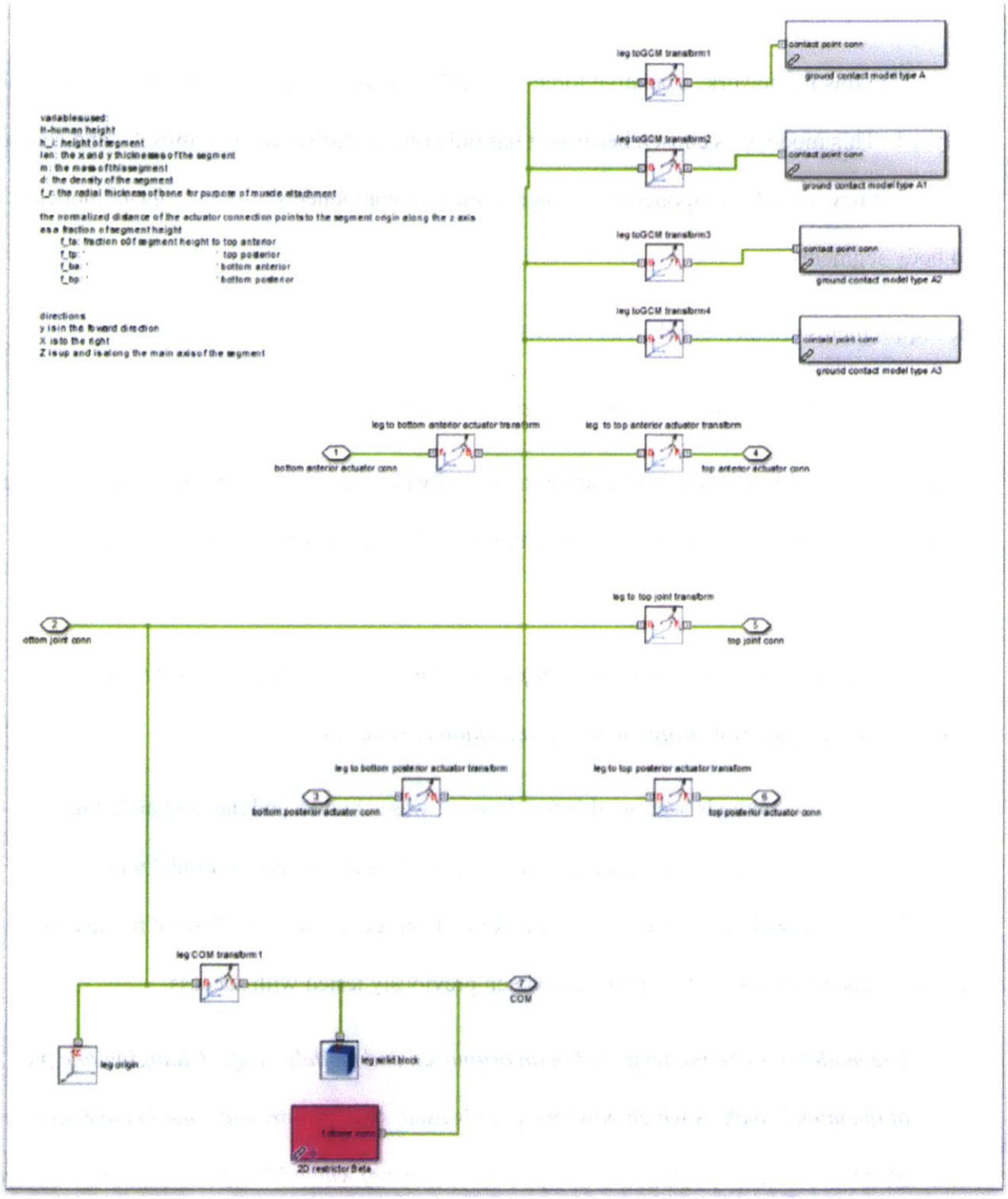


Figure 19 shows the canvas for the implementation of the universal leg segment. This segment contains ground contacts, mappings to attachment points, and a restrictor for two dimensional movement.

Figure 19: Universal Leg Segment Implementation



Evaluation and Results

To evaluate this framework, a positive force feedback hopping model [24] was implemented and analyzed. This model was chosen because it has only one controller and one muscle, and can be used to test new muscle components and energy analysis components, ground contact models, and body segments.

The requirements are then checked as follows:

1. *The simulation shall be capable of closed loop feedback control.*

The controller used in this test is a positive force feedback controller which takes a delayed force signal from the muscle and outputs a stimulation signal. This requirement was successfully fulfilled.

2. *The optimization shall evaluate multiple simulations in parallel to reduce the time the developer spends waiting for the optimization to complete.*

As described in the methods section, this requirement was achieved and then expanded upon. The model was successfully debugged in configurations V and VI, and the model was successfully optimized in configurations III and IV. Configurations I and II were not used with this model due to operational cost, but have been previously tested with success.

3. *The model shall be implemented in an organized and modular way. Changing one part of the model, such as which muscle model is used, shall require only one implementation of the new muscle model, and a simple substitution of which library components are used for each relevant muscle in the model.*

This model was implemented with reusable library components. Modifications to these components occur in the library, and parameters are applied at the top level of the Simulink model only. This requirement was successfully fulfilled.

4. *The main body of code that optimizes the model parameters shall be simple, well organized, and reusable across different models.*

The main body of code is brief, and has been altered to suit other models such as the Endo-Herr walking model.

5. *There should be an accumulating library of reusable model components, such as body segments, muscles, ground contact models, and controllers. Each time a model is implemented, this library is increased in size. Each time a library component is improved in efficiency or organization, the change is persistent and spreads to others who are using the same library.*

The model components are suitable for this Simulink library. In addition, during this evaluation, an error in our metabolic cost library component was discovered. This error was corrected in the library component, and was successfully extended to the model, and to the components of the Endo-Herr implementation under development.

6. *There shall be an accumulating archive of successfully running models that can used as a starting place for building new models, so that each new component can be tested within a whole system.*

This model implementation counts as the first archived model. This requirement is successfully fulfilled.

7. *The code used for the optimization shall be sourced from a library with the choice of machine learning algorithm being as simple as changing a few lines of code. Implementing a custom genetic algorithm shall be unnecessary.*

A wrapper function around the Matlab toolbox genetic algorithm function sets all of the input parameters for the optimization process. We also created a similar wrapper function around the Matlab toolbox particle swarm optimization.

8. *There shall be a highly customizable analysis mode that can run simulations using pre-tuned parameter sets and produce graphic videos of the gait, and any desired plot of simulation state information like force, torque, etc.*

The analysis mode has turned out to be more of a simple script that loads the data and runs the simulation. Data analysis is one of the areas where Matlab shines. Leaving our analysis script open to modification allows the researcher to freely measure the simulation. The analysis mode is also able to run the Simulink model in real time, with video on. This allows us to graphically depict the animation and the internal state of the Simulink blocks simultaneously. This is useful for models with complex state machine controls. Analysis that requires information from each timestep, such as metabolism, spring energy, or maximum height is computed during simulation to eliminate the need for saving massive vectors for post processing in Matlab. This requirement is successfully fulfilled.

Simulation Analysis

This implementation is distinguished from the model described in the original paper in several ways. First, the model is built using body segments that are based on anthropometry measurements from [22], [23]. In our implementation, the legs have mass and inertia, based on a

block-like shape. The range of motion of the end effector on the lower leg segment is constrained to move vertically below the HAT segment, and the HAT segment is also constrained such that it may only translate in the vertical direction.

The addition of leg mass is important because we are able to show that even in the absence of a controller for leg position, hopping is still possible. Hopping apex return maps were generated, along with video animations that contain concurrent energy and control state information.

The eventual goal of this system is to build up a selection of implementations of various models of gait, including walking and running. In the spirit of modularity and incremental design, the first model reproduced with this system was a force feedback hopping model, based on the 2003 paper by Hartmut Geyer [24]. This model is a simple gait model with one reflex and one muscle. It serves as a reliable platform for testing and proving new ground contact, body segment, analysis, muscle, and control modules.

The original model is a two segment leg model with all of the body mass centered at the hip joint. The legs are massless, and the muscle attachment is virtual, with the shortening of the MTU based on the angle of the knee.

In our reproduction of the model, the muscle attaches to specific coordinates on the leg segments, and the leg segments have mass. The ground contact is based on a simpler spring-damper model, and the entire model is constrained to translate in the vertical direction. In this way, we are able to test the viability of this model to maintain stable hopping with added leg mass.

Using the weight as described in the paper, a model was built with a (Head, Arm, Torso) HAT segment, a thigh segment, and a shank segment, with sizes based off of the textbook anthropometry and mass distribution [22].

The model described by Geyer did not have mass in the legs, but the implementation described as follows does. Because the legs have mass, the results should differ from the original paper in some way. In addition, now that the legs have mass, the state of the simulation is not completely represented by the hip height return map, as legs can be in different positions at the apex.

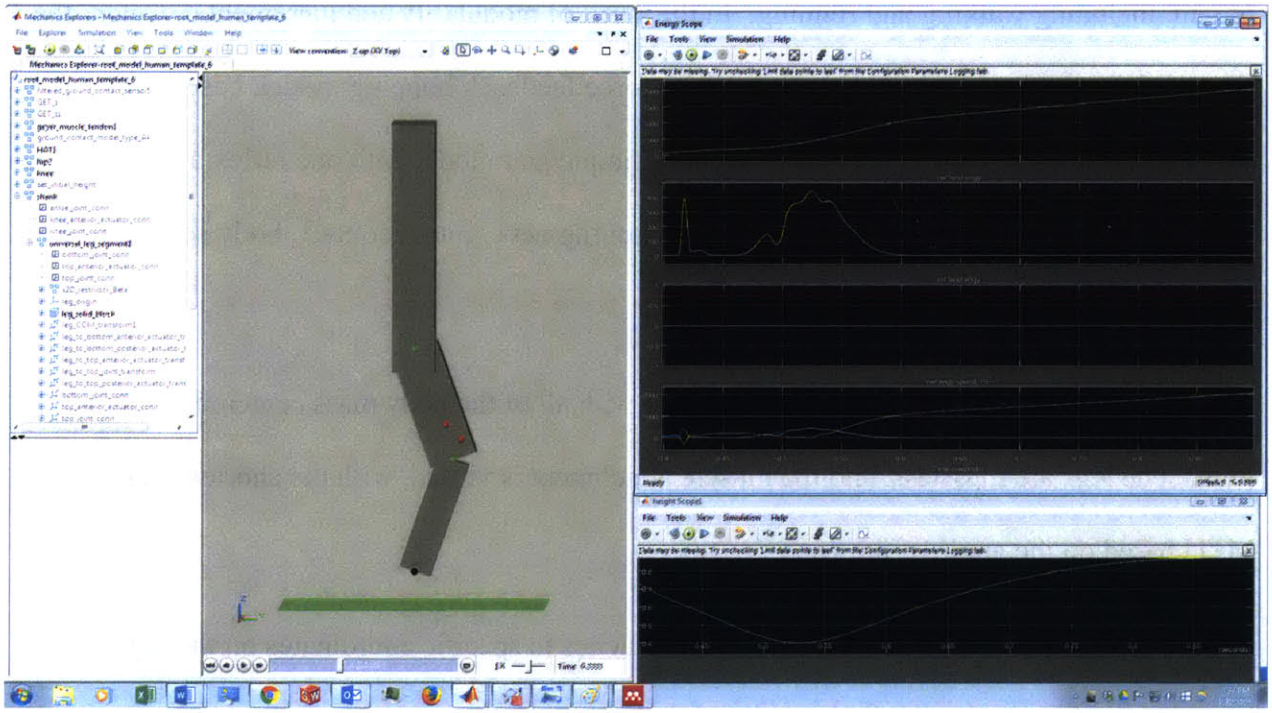


Figure 20: Simulating with Animation and Scopes

The following six variables were optimized, the ranges shown in their axes:

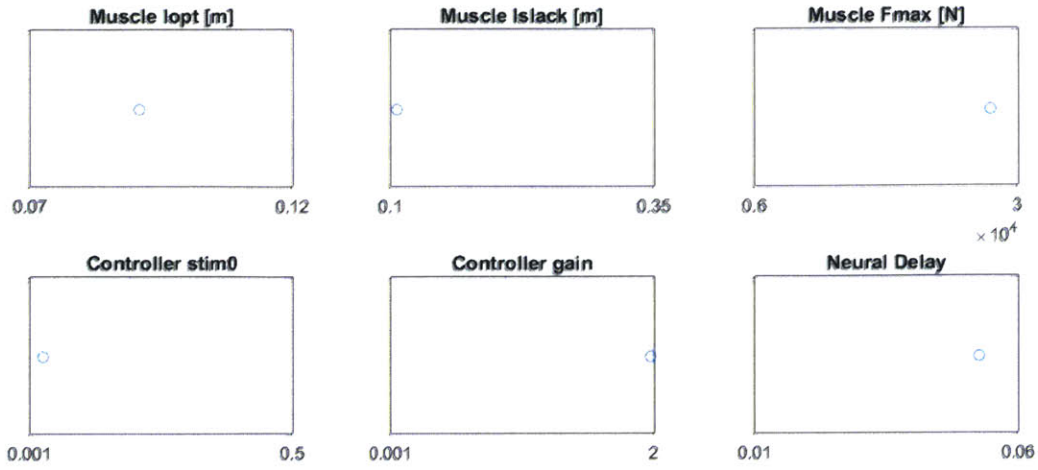


Figure 21: Optimized Parameters

The optimization produces a recurring hopping pattern, but the return map does not show the same convex shape as the original paper. This is likely due to the addition of leg mass, and the ability of the legs to change angle according to body dynamics while in the air, two features not included in the original model.

The following plot shows the return map of the model. Height is recorded at the hip, at the apex of the hop.

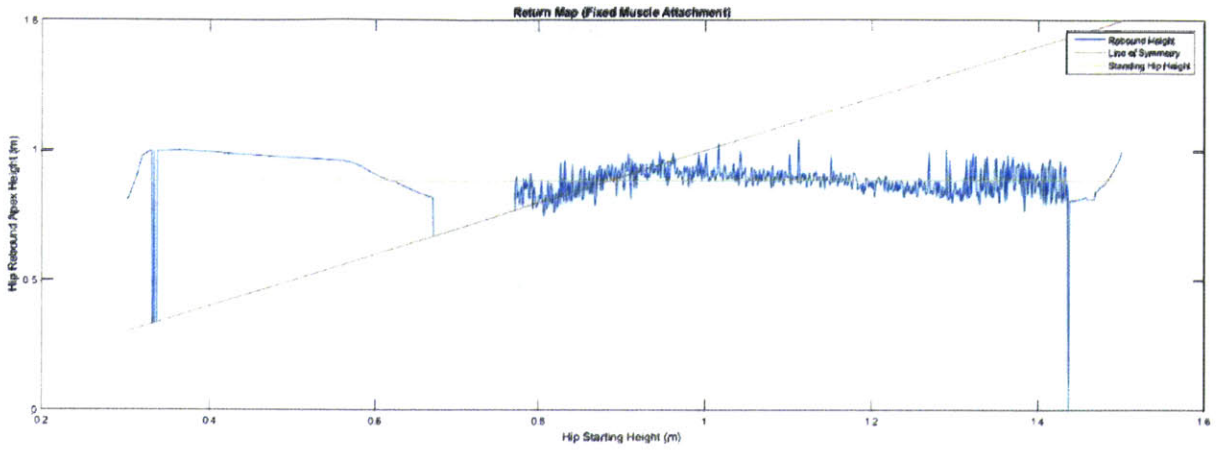


Figure 22: Return Map Using Fixed Muscle Location

We then added more parameters to the optimization to see if hopping height could be increased. This resulted in less stable hopping, but a much higher peak height. The optimization was halted early once the heuristic showed higher hopping. This is why a broader range of parameters exists. Adding more parameters did not increase the stability of the hopping with the original heuristic. When tuning parameters that can stretch the muscle past its slack length at initialization, it was necessary to use energy analysis in the tendon to prevent pre-stored energy from increasing hop return height. This was accomplished by computing spring energy in the tendon at initialization, converting that to a potential energy height from gravity that was added

to the drop height in the computation of the heuristic. In this way, the optimization does not select for pre-loading the tendon.

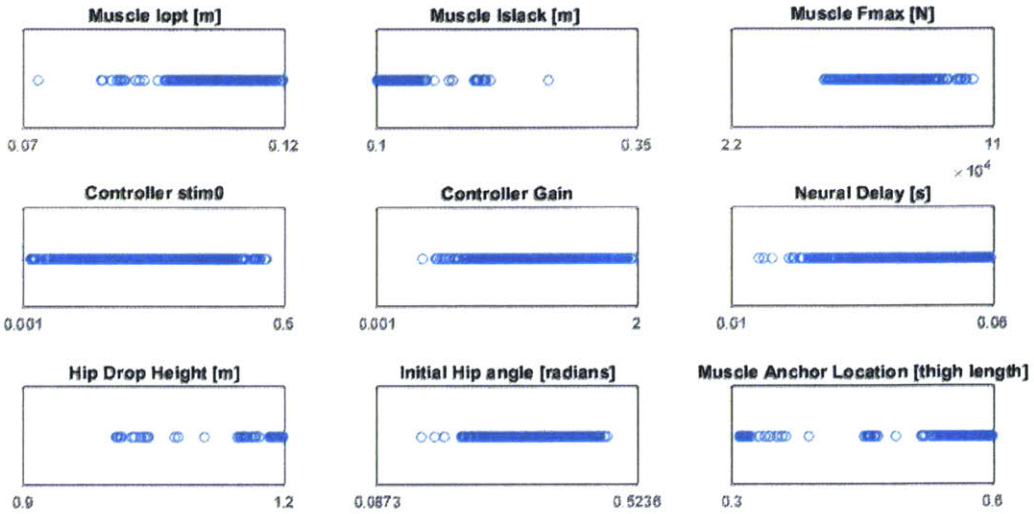


Figure 23: Optimized Parameters Trial 2

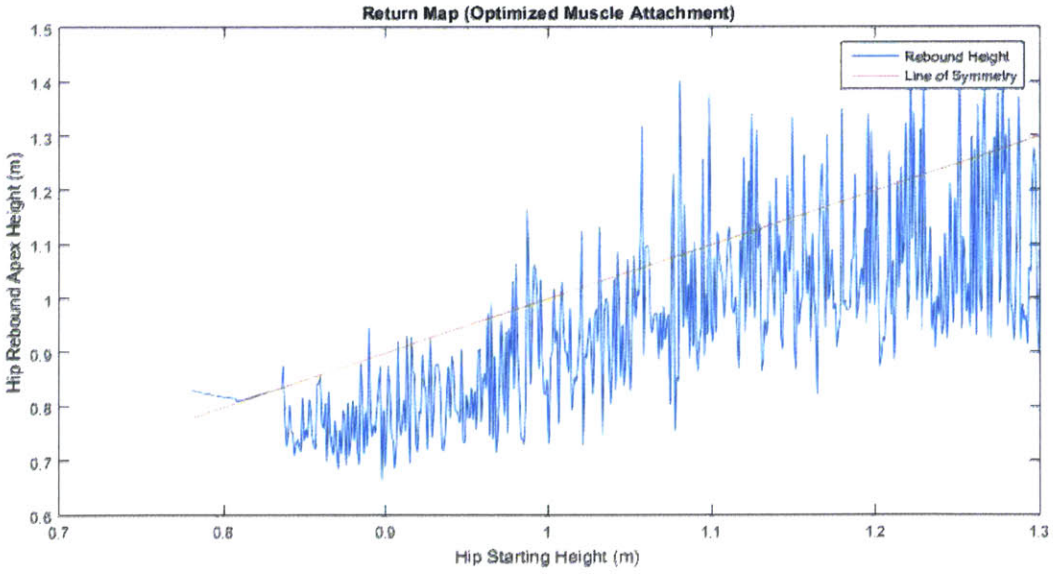


Figure 24: Return Map Trial 2

By using a real-time pacer to slow down the Simulink simulation, it is possible to render an animation in time with Simulink scopes and control state animation. This video (Figure 25) was

produced by running a simulation and recording the animation using Open Broadcaster Software.

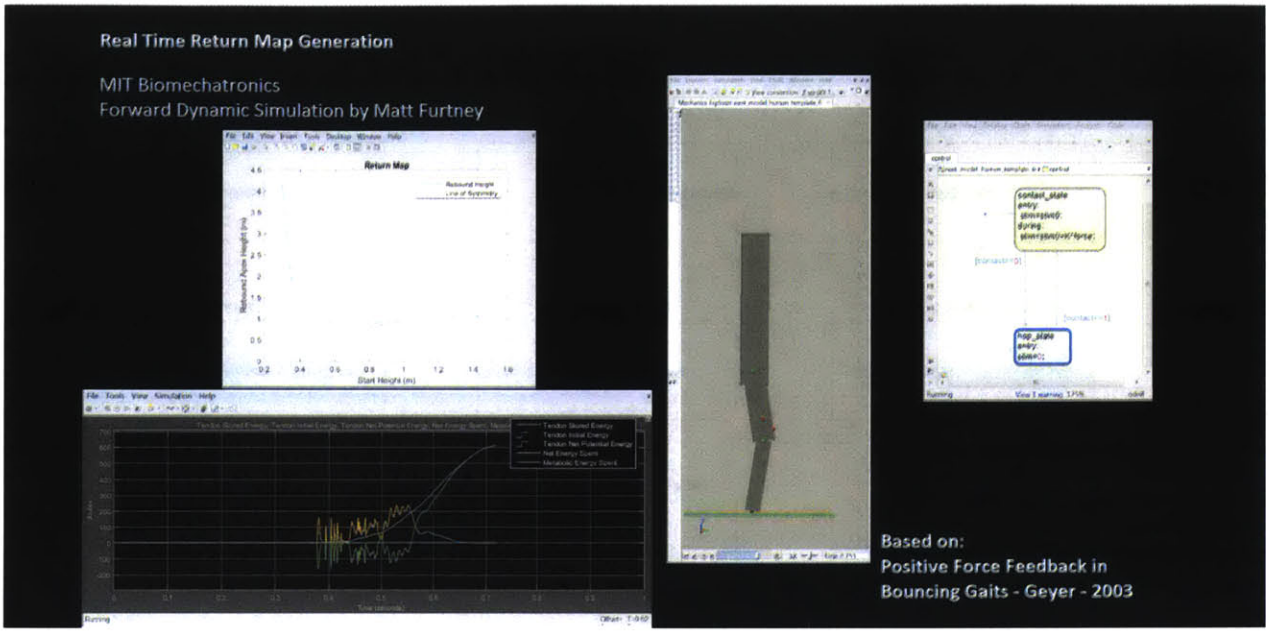


Figure 25: Demonstration Video Screenshot

Figure 25 shows a screenshot of a demonstration video, which illustrates the return map generation, while animating the mechanics, controls, and muscle energy plots.

Conclusion

At the beginning of this project, I set out to improve upon a well-established gait model by iteration. In doing so, I discovered a great challenge in bridging the gap between successive iterations on forward dynamic models, and in the reproduction of models based on their description in academic journals. This challenge inspired me to design a system that could bridge the gap between model conception and model testing in a way that accelerated the pace of discovery. Once implemented, this system was able to fulfill our original requirements. Our initial results show promise in the acceleration of gait model development.

References

[1] R. M. Alexander, *Principles of Animal Locomotion*. Princeton, NJ, 2002.

[2] V. A. Tucker, "The Energetic Cost of Moving About," *Am. Sci.*, vol. 63, no. 4, pp. 413–419, 1975.

[3] L. M. Mooney, E. J. Rouse, and H. M. Herr, "Autonomous exoskeleton reduces metabolic cost of human walking during load carriage.," *J. Neuroeng. Rehabil.*, vol. 11, no. 1, p. 80, May 2014.

[4] "Hardiman I Arm Test: Hardiman I Prototype Project," Schenectady, New York, 1969.

[5] H. M. Herr and a. M. Grabowski, "Bionic ankle-foot prosthesis normalizes walking gait for persons with leg amputation," *Proc. R. Soc. B Biol. Sci.*, vol. 279, no. 1728, pp. 457–464, Feb. 2012.

[6] G. H. Traugh, P. J. Corcoran, and R. L. Reyes, "Energy expenditure of ambulation in patients with above knee amputations," *Arch. Phys. Med. Rehabil.*, vol. 56, no. 2, pp. 67–71, 1975.

[7] R. H. Miller, "A comparison of muscle energy models for simulating human walking in three dimensions.," *J. Biomech.*, vol. 47, no. 6, pp. 1373–81, May 2014.

[8] S. Coros and B. Jones, "Locomotion Skills for Simulated Quadrupeds," pp. 1–11, 2011.

[9] 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," vol. 54, no. 11, pp. 1940–1950, 2007.

[10] T. Dimensions, "Computer Methods in Biomechanics and Biomedical Engineering," no. 764699363, 1999.

[11] T. Geijtenbeek and A. F. Van Der Stappen, "Flexible Muscle-Based Locomotion for Bipedal Creatures."

[12] S. Song and H. Geyer, "A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion," pp. 1–32.

[13] MathWorks, *Simulink User's Guide R 2015 a*, 2015 a. 2015.

[14] Mathworks, *Simulink Reference 2015 a*. 2015.

[15] MathWorks Automotive Advisory Board, *Control Algorithm Modeling using Matlab, Simulink, and Stateflow*, 3.0 ed. .

[16] MathWorks, *Stateflow*, 2014th ed. .

[17] N. Marshall and S. Flight, "Simulink Code Generation," 2012.

[18] K. Endo, D. Paluska, and H. Herr, "A quasi-passive model of human leg function in level-ground walking," in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 4935–4939.

[19] H. H. Ken Endo, "A model of Muscle Tendon Function in Human Walking," *2009 IEEE Int. Conf. Robot. Autom.*, no. October, 2009.

[20] K. Endo and H. Herr, "A model of muscle-tendon function in human walking at self-selected speed," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, no. 2, pp. 352–362, 2014.

[21] K. Endo and H. Herr, "Human walking model predicts joint mechanics, electromyography and mechanical economy," *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, pp. 4663–4668, 2009.

[22] D. A. Winter, "Biomechanics and Motor Control of Human Movement: Fourth Edition,"

2009.
[23] J. W. Rohen, C. Yokochi, and E. Lutjen-Drecoll, *Anatomy: A Photographic Atlas*, 8th ed. Wolters Kluwer, 2015.
[24] H. Geyer, A. Seyfarth, and R. Blickhan, "Positive force feedback in bouncing gaits?," *Proc. Biol. Sci.*, vol. 270, no. 1529, pp. 2173–2183, Oct. 2003.