The Dissertation Committee for Joseph Lindell Cooper
certifies that this is the approved version of the following dissertation:

Analysis and Synthesis of Bipedal Humanoid Movement: A Physical Simulation Approach

Committee:

Dana H. Ballard, Supervisor

Donald S. Fussell

Richard R. Neptune

Luis Sentis

Peter Stone

Analysis and Synthesis of Bipedal Humanoid Movement: A Physical Simulation Approach

by

Joseph Lindell Cooper, B.S.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2013

Analysis and Synthesis of Bipedal Humanoid Movement: A Physical Simulation Approach

Publication No. _____

Joseph Lindell Cooper, Ph.D.
The University of Texas at Austin, 2013

Supervisor: Dana H. Ballard

Advances in graphics and robotics have increased the importance of tools for synthesizing humanoid movements to control animated characters and physical robots. There is also an increasing need for analyzing human movements for clinical diagnosis and rehabilitation. Existing tools can be expensive, inefficient, or difficult to use.

Using simulated physics and motion capture to develop an interactive virtual reality environment, we capture natural human movements in response to controlled stimuli. This research then applies insights into the mathematics underlying physics simulation to adapt the physics solver to support many important tasks involved in analyzing and synthesizing humanoid movement. These tasks include fitting an articulated physical model to motion capture data, modifying the model pose to achieve a desired configuration (inverse kinematics), inferring internal torques consistent with changing pose data (inverse dynamics), and transferring a movement from one model

to another model (retargeting). The result is a powerful and intuitive process for analyzing and synthesizing movement in a single unified framework.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation presents research into capturing, analyzing, and simulating bipedal humanoid movements. A virtual reality environment developed as part of this research supports controlled experiments for systematically recording human behaviors. Specifically, we use virtual reality in combination with multiple sensors to record human behaviors in response to natural situations under scientifically controlled conditions. After capturing behavior data, we leverage numerical simulation, using robust, open source software libraries to process the data. The objective of this dissertation is to develop a method for using physical simulation software as a tool for analyzing and synthesizing humanoid movement with sufficient speed and accuracy to allow the analysis and synthesis to be used for real-time interactive applications such as psychophysics experiments using virtual reality or human-in-the-loop teleoperation of a simulated robotic system. Achieving this objective requires that our techniques be fast and robust while still providing results sufficiently accurate to be used to believably animate a humanoid character, control a simulated system, or estimate internal joint forces used during a movement for creating effort-contingent experimental stimuli.

The simulation libraries used in this research, originally designed to provide

physical simulation for game worlds, serve as a powerful tool, not only for modeling the movement of a simulated character, but also for analyzing recorded movement data and creating new, goal-directed movements based on human data. Although other tools and approaches exist for analyzing and synthesizing movement, the techniques developed as part of this research provide a unified approach for solving a multitude of tasks in a way that is fast, intuitive, robust, and free.

The novel, simulation-based approach to analyzing and synthesizing movement presented in this dissertation uses forward-dynamics simulation to extract useful kinematic and dynamic measurements from human data. The unifying theme of this dissertation is to capture behavioral data directly from humans and then use the Open Dynamics Engine (ODE), a software library designed for simulating physical systems, to (1) fit a physical model to captured data, (2) extract model pose from the data, (3) adapt the model pose to satisfy new constraints, and (4) quickly compute force/torque estimates that produce a given pose sequence in a simulated model. These tasks are play prominent roles in animation, robotics, and biomechanics.

1.1 Motivation

Human-like characters are used extensively in games, movies, and other domains employing animation. Creating believable, task-directed movement in human-like characters is typically a labor-intensive process and often still results in movements with unpleasant visual artifacts. Forward physical simulation for animation has become increasingly possible as a preferred alternative to hand-crafted keyframe methods. Data-driven methods using motion capture data[38] or model-based meth-

ods projecting a low-dimensional controller onto a high-dimensional character[74] can be used to convincingly animate humanoid characters, but several steps are required to integrate these approaches with forward physical simulation. Adapting the software used for forward simulation to handle many of the steps necessary to analyze motion capture data or project movement from a low-dimensional model up to full-humanoid character simplifies the integration process.

Much like simulated characters, humanoid robots are becoming increasingly commonplace. Many of the techniques used to animate graphical characters can be applied to control physical robots. Intelligent but simple methods for simulating, understanding, and controlling robots are growing in relevance. For example, rapid inverse dynamics computations may facilitate the use of advanced control techniques such as impedance control [47] where a mechanism responds to external forces as though it had physical properties different from its actual dynamics. Impedance control requires evaluating the effects that external forces would have on a system with the desired dynamic properties and then computing appropriate control signals for realizing those effects in the actual robot. A physics-based approach naturally lends itself to this type of application where analysis and synthesis of movement operate alongside real-time forward simulation. With the right tools for analysis and synthesis, it is possible to leverage human learning and insight to solve complex control problems. Increased understanding of the computation underlying human movements can inspire the control algorithms driving robotic agents.

In addition to graphics and robotics applications, measurements of human activity are useful for studying the neural computations underlying human behav-

ior. Dynamic models of human behavior also support clinical efforts to analyze, rehabilitate, and predict movements. Dynamic models are used in biomechanics to understand and diagnose motor pathologies, to find new motor strategies that decrease the risk of injury, and to predict potential problems that might arise from a particular procedure[88]. Although for many clinical applications, measurement accuracy is more important than speed, real-time analysis of human joint torques can be used for biofeedback-based treatments. Rapid and robust analysis of movement also makes it possible to conduct interactive experiments that rely on an estimate of the forces underlying a movement as an approximation of human effort or 'movement cost'.

Many methods exist for finding pose based on motion capture marker data or computing torques and forces from pose data. However, standard approaches can be difficult to integrate into a real-time application. Commercial packages may be expensive, inflexible, or confusing. Specialized approaches for dealing with the various stages of movement analysis may be difficult to combine, making use of different models, standards, and formats. Some movement analysis tasks, such as inverse dynamics, are notoriously slow processes. An inexpensive, efficient, and robust approach built into a single framework can facilitate movement research and improve the process of synthesizing movements for robots or animated characters. This research represents a significant advance toward that goal.

The research in this dissertation uses virtual reality with motion capture to acquire data on human behavior and then adapts physical simulation software to accomplish important kinematic and dynamic analysis. Virtual reality with motion

capture serves as a method for capturing full-body movements during natural tasks while providing powerful experimental control over stimuli perceived by a human subject. Direct control over a simulated environment allows researchers to manipulate the reality perceived by a human subject as necessary to study the human response. Virtual reality also increases the number of experimental trials possible in a limited amount of time by making it possible to automatically reset experimental conditions as soon as the subject is ready to continue.

Interactive experiments require that visual stimuli change in real time in response to the human movements. The optical motion capture system used in this work provides the 3d spatial coordinates for a set of active markers attached to a suit worn by the human subject. The marker coordinates provide information about the subject's pose and viewpoint within the world. Analyzing marker information in order to match the human pose with a simulated model is a key step in using virtual reality to study human movement. A real-time method for finding human pose from motion capture data allows a researcher to design experiments with pose-contingent stimuli and can provide a human subject with an avatar, a representation of his or her body within the virtual environment.

A physics-based approach to analyzing marker data makes it simple to extract character pose from marker data and then use that pose to control a simulated character in real-time in a virtual reality experiment. Employing a single physics-based framework for forward simulation as well as movement analysis and synthesis in a virtual world means that less code is involved and there are fewer potential conflicts involved in going from captured data to an animated result.

1.2 System Complexity

Although humanoid systems are extremely complex, humans are capable of controlling their own movements flexibly and dynamically to accomplish many different tasks. Precisely how humans accomplish what they do is poorly understood but is interesting to many disciplines from biomechanics and neuropsychology to robotics and animation. Understanding the kinetics and kinematics of humanoid systems is desirable for many reasons but is challenging because it is a domain of high-dimensional spaces with non-linear interactions.

Virtual reality provides the opportunity to study human movements in conditions that are more natural than traditional laboratory settings. Full-body motion capture supports the study of free and natural movement, but this freedom brings complications. Studying different features of natural behaviors can involve many different types of input and output devices for generating appropriate stimuli and capturing responses. Integrating multiple devices and software libraries to create an immersive experience for scientific inquiry is non-trivial.

Tools for analyzing and synthesizing human movement must deal with the complexity of the underlying system. A crushing number of variables are involved, even for brief behaviors. Applying computational methods to the problem of studying behavior requires discretizing continuous temporal data into a sequence of T frames over time. Handling discretized data introduces some challenges. More challenging still is the variability of the human body. Humans have many bones, joints, and muscles that provide hundreds of degrees of freedom. The various human body segments vary in size and mass. The space of possible body poses is incredibly large. Even

our simplified model still uses 54 degrees of freedom in order to capture important features of movement.

This complexity comes into play when trying to find a model pose the matches the data at a specific point in time. Optimization approaches in this space are rather costly in terms of time and computation. The problem is aggravated by the fact that optical markers attached to the skin tend to shift around relative to the bones and articulation points, violating assumptions of rigidity. Moreover, tendons and ligaments allow bones to move and separate slightly during movement, causing problems for a simplified model that assumes fixed centers of articulation.

Finally, the problem of finding forces and torques that move a character model from one pose to another is generally an ill-posed, two-point boundary value problem. Human data become particularly useful in addressing this challenge. The non-linear humanoid model would be very difficult to analyze and control over large windows of time, but the human data give critical information that enable a relatively simple linear system to recover the dynamics on a per-frame basis.

1.3 Principal Contributions

This dissertation touches on multiple disciplines and has produced several useful software applications and computational techniques. The overall goal of this work is to provide a real-time, physics-based mechanism for obtaining, analyzing, and synthesizing humanoid movement data in a unified framework using intuitive parameters and fast, computationally efficient methods. We accomplish this goal by adapting freely available software libraries—designed for robust, real time physical

simulation—to solve multiple problems associated with modeling human movement, eventually producing estimates of joint torques and forces that can be applied to animate simulated characters, control physical robots, or estimate joint strain. This mechanism leverages code already needed for forward simulation to accomplish these tasks. The principal contributions of this work are summarized here:

- The virtual reality interface developed during the course of this work integrates multiple sensors and devices and is being used for interactive experiments exploring human behavior in simulated natural environments.

- A clear explanation of how the equations of motion are applied in physical simulation leads to several insights on how the robust solver for constrained dynamical systems can be used for multiple purposes such as fitting a character model to real-world marker data, adjusting model pose to satisfy positional constraints (inverse kinematics), and computing torques necessary to achieve a target pose (inverse dynamics).

- Methods for accomplishing these different tasks are described and validated using an example implementation.

- The process of applying these methods involved deriving and implementing multiple useful additions to the freely available Open Dynamics Engine (ODE) project to improve stability and add flexibility. This physics simulation software is used in multiple communities as a research tool and improvements to it can have a broad impact.

These novel contributions are useful for animation, robotics, biomechanics, and psychophysical experiments.

1.4 Organization

The organization of the remainder of this dissertation is as follows:

Chapter 2 presents a general sampling of topics and recent research relevant to the contributions in this paper.

Chapter 3 covers our work in creating a virtual reality environment that incorporates multiple types of sensors and provides real-time audio-visual feedback for an immersive experience. This work is useful for producing controlled conditions that allow systematic capture of human movements and associated data, conditioned on a specific task. This chapter also includes a brief discussion of experiments made possible by this virtual reality system.

Chapter 4 gives an overview of physical simulation. We discuss the use of different numerical representations of constraints used in physical simulation. We present a detailed discussion of the algorithms and mathematics underlying the simulated physics at the core of this research. This understanding is then leveraged to apply the constraint solver to other purposes such as inverse kinematics and inverse dynamics computations.

Chapter 5 focuses on using simulation software to analyze motion capture data. We demonstrate how to use the physics engine to extract useful information from data recorded from motion capture and other sensors. This chapter includes

methods for fitting a model to the data and then finding control signals (torques) that produce the observed data.

Chapter 6 presents several ways to employ the simulation software to facilitate common tasks in movement synthesis and control for animation and robotics. Among these tasks are methods for retargeting captured data to a new model and applying novel constraints to achieve desired movement properties.

Finally, in Chapter 7, we summarize the contributions of this research and present directions for future work.

Chapter 2

Related Work

This dissertation combines virtual reality with numerical simulation techniques to accomplish a number of different goals that are relevant to many different fields. Consequently, the body of related research is particularly large and multidisciplinary by nature. One of the principal contributions of this work is a fast, robust method that takes motion-capture marker data and eventually produces internal joint torques. Along the way to computing torques, we must find a physical model that fits the data along with a mapping that associates markers with corresponding points on the model. Fitting the model to a particular frame of marker data gives a target pose. Finally, with that information we can compute the joint torques that take the model from one pose to a target pose over a short time step. The result is a single, multi-purpose, physics-based tool for accomplishing many important tasks related to movement analysis and synthesis.

We are unaware of any other demonstration of a single framework being used for all the different tasks covered by physics-based analysis and synthesis. The different sub-tasks in this process, each with its own body of literature, have various applications for the fields of animation, robotics, and biomechanics. In some cases, the different fields use conflicting terminology. In this chapter, we will touch on liter-

ature from these different fields as it relates to virtual reality, constrained rigid-body dynamics, movement analysis, and movement synthesis.

2.1 Virtual reality

Our work constructing a virtual environment as a testbed for psychophysical experiments is not especially novel, but remains worthwhile contribution. It is also the first place that we employ physics-simulation as a tool. However, in this case, we primarily use it as it was intended to be used. Virtual reality has been used extensively for entertainment, training, and rehabilitation[14]. It is somewhat less common to use it for studying human movement, but some examples do exist[41, 82]. We present our work developing a virtual environment for studying human movement here because of its usefulness to the overall physics-based technique that is the core of our research.

2.2 Rigid-body dynamics

There are two main approaches for modeling the equations of motion of an articulated model: generalized (or reduced) coordinates and maximal (or Cartesian) coordinates[59]. These two approaches are complementary, in a way. At a high-level, the distinction between these two approaches is that generalized coordinates methods add degrees of freedom to a system by introducing articulations while maximal coordinates methods take away degrees of freedom by applying constraints. When animating or analyzing humanoid characters, it is common to use generalized coordinates[71]. General purpose rigid body simulation typically employs maximal coordinates. We

use maximal coordinates as implemented in ODE to simulate and analyze articulated humanoid characters.

Generalized coordinates represent an articulated body only by its degrees of freedom. For example two bodies connected by a hinge can be represented by the position and orientation of the first body and the hinge angle because the orientation and position of the second body are fully determined by that information. The state of the second body is implicit from the hinge state. In this way, degrees of freedom are explicitly added to a system. Typically, a character model using generalized coordinates is constructed hierarchically with one body defined relative to the world and all other bodies defined by a chain of linkages to this root node[36]. Generalized coordinates typically result in a low dimensional representation. However, it must be possible to describe the full state of the body as a function of generalized coordinate space. This requirement also makes it difficult to represent "kinematic loops" such as when both feet are on the ground and for large systems, the equations of motion in nested, rotating reference frames become very complex, making them more difficult to approximate well.

In contrast, maximal coordinates place each rigid body in the global frame with six degrees of freedom. Constraints removing degrees of freedom from the system are added explicitly, leaving the bodies to move freely where unconstrained. Although for character animation and biomechanics research, generalized coordinates seem to be more common[71, 116], simulation in this research occurs in maximal coordinates. We use the Open Dynamics Engine (ODE)[99] which came out of Smith's doctoral work[98]. The physics computations in this library are based on early work

by Baraff[8], but have been augmented by a community of interested programmers and hobbyists.

Rather than represent the body state as a function of generalized degrees of freedom, we must be able to represent constraint state as a function of the body state. Because the joints connecting body parts are not implicit, numerical errors and other factors can allow connected body segments to separate. Though it may seem advantageous to keep body parts rigidly connected, in real human bodies, joints, tendons, and ligaments stretch. Describing a system with maximal coordinates also makes models easier to build because the constraint state typically only relies on the the state of one or two bodies, whereas a computing the position of a from a set of joint angles along the arm requires describing all relationships up the chain. Framing joint constraints this way can decrease problems with error propagation on long chains of bodies, allows kinematic loops, and has other strengths. Taking advantage of these strengths is key to this dissertation.

As we have worked to augment the constraints available in the physics simulation code, uncontrolled manifold (UCM) theory[91] inspires much of our approach. This theory observes that multiple trajectories may accomplish a task equally well. Rather than choose a single trajectory and fixedly try to follow it, people will take no corrective movement if a perturbation leaves the movement on an acceptable path. Many experiments supporting this hypothesis are presented in [68] and [95]. It is interesting to look at these results from human data as potential inspiration for our system. A much more thorough review specifically focused on computational motor control is available in [55]. We have created a novel set of partial constraints that can

be used to imitate this type of control.

2.3 Movement analysis

The principal tasks we are interested in when analyzing movement data are fitting a humanoid model to capture marker trajectories (finding the appropriate body dimensions and pose) and then finding joint torques that produce the appropriate movements.

Related work for building a model automatically constructs a kinematic skeleton completely from marker data[26, 58]. Although convenient, such approaches may not find the desired segments and linkages. In particular, automatic methods cannot find body segments that do not have any markers on them. For example, we do not place any markers on the neck still treat it as a separate segment because doing so allows the head to move with five degrees of freedom relative to the torso. Model building approaches frequently find the size and pose of the skeleton simultaneously by using non-linear optimization to minimize disagreement between markers and model segments. We propose a similar approach that uses the physics engine to handle a major part of the work.

Zordan and Horst presented the technique which is most similar to at least part of our work[125]. Their work situates motion capture markers in the simulated space and assigns them to attachment points on a character model. They then compute spring/PD forces that they apply to the model, pulling the body toward the markers. The principal difference between this method and our approach for fitting a model to marker data is that we allow the physics engine to handle the springs as constraints,

making the process faster and more stable. The idea of dragging simulated bodies into place with constraints was actually proposed long ago by Barzel and Barr[9]. However, the technique was challenging in practice because of singular constraint systems. Softened constraints that behave like springs get around this problem and make it possible to still solve over-constrained systems and pull a model toward a set of kinematic targets. The same approach also works for finding internal joint forces and torques (inverse dynamics).

Standard approaches for computing inverse dynamics in the literature describe a humanoid model using generalized coordinates and then compute the forces by inverting the effective mass (or finding a pseudo-inverse if ground-reaction forces are known)[64]. Generalized coordinates represent a character model with a hierarchy of joint angles, treating joint constraints themselves as implicitly satisfied (holonomic). There are a few problems with this approach. The first problem is that the tree representation dictates that exactly one foot (the "root node") is assumed to be stationary (rigidly attached to the ground). Having more than one foot on the ground results in a kinematic loop. Having no feet on the ground leaves the model underactuated. Closed kinematic loops cause problems for physics formulated in generalized coordinates as do underactuated degrees for freedom. This fixed foot can typically produce constraint forces in both directions (rather than only away from the surface). Another problem with the generalized formulation is that human joints are not holonomic. Tendons flex and give as the body moves as does the body surface to which markers are attached. Dealing with these "soft tissue artifacts" is a significant issue in biomechanical analysis of movement[69]. Working in maximal coordinates with soft

constraints provides a natural way for allowing some natural stretching and flexing to happen.

The creators of OpenSim recently described a process taking from several days to several months to produce a reasonable simulation[86]. This computational cost serves as ample justification for employing a faster method. OpenSim, a free software package aimed at biomechanical analysis, models movement in generalized coordinates which typically involves creating subject-specific simulation code. The same paper laments the lack of open-source tools for modeling and simulation biomechanical systems. We seek to help fill this niche.

Standard biomechanics methods use an abundance of markers to track each rigid body, rendering models largely unnecessary. Soft tissue artifacts deform the "rigid" arrangement of markers, making it so that model fitting techniques must minimize "fit-error". In one study, adding knee joint constraints did not reduce error[3]. Our technique works with far fewer markers. With few markers, modeling joint limits and stiffness contributes significantly.

Remy and Thelen[88] describe what is approximately the state of the art in inverse dynamics computation for biomechanics analysis. This method aims to constrain the system to balance body accelerations with measured ground forces, gravitational forces, and gyroscopic forces. Although the paper asserts that the method eliminates the need for residual forces, one of the key steps is to allow "small variations" in the generalized accelerations as well as the ground forces and torques. Although this approach shifts the residual forces from the waist to the feet, they are still there. It seems that no steps have been taken to ensure that the ground does

not "pull" the body unrealistically downward. Because the method changes body accelerations, the pose that results from numerically integrating the accelerations will differ from the measured pose obtained through motion capture and inverse kinematics. The authors use numerical optimization to find initial conditions (generalized positions and velocities) that lead to the best agreement with the final conditions when coupled with the new accelerations. The method presents two error metrics: agreement of pose and agreement of marker position (optionally weighted). The goal of the process, then, is to minimize the integrated error of the combined metrics over an entire movement. We will employ a similar metric in analyzing our approach in Chapter 5. The approach is very good, but runs slowly and still stands to introduce certain inaccuracies in the results. In particular, low-pass filtering and spline fitting risk smoothing away the high frequencies that accompany ground collisions. This optimization approach also reports computation times of approximately 100 minutes for processing a single gait cycle. The inverse dynamics technique described in this dissertation accepts residual forces as part of the computation and sacrifices some accuracy in return for speed; however, we also present, as future work, methods for reducing residual forces and improving accuracy without adding excessive computational cost. Our technique runs in real-time and can even process multiple models simultaneously, making it suitable for applications requiring analysis at interactive rates.

2.4 Movement synthesis

Early use of physical simulation for creating new movements was largely limited to rag-doll effects. Under this paradigm, kinematic trajectories completely define the behavior of a simulated character until an impact or significant event occurs. Typically that event involved the character's death because responsibility for animating the character would be passed to the simulation engine which would then fling the now-floppy character around according to physical laws. Because of the difficulties involved in controlling humanoid characters. Early work in this area controlled the lower body kinematically while tracking reference motion for the upper body with proportional-derivative (PD) control (cf. [122]). This type of system was used to generate animated boxing and table-tennis agents[123, 124]. This same strategy is available in the framework we describe, but it is possible to do much more.

Motions graphs[6, 62] are common structures used to store and synthesize movements. This structure formalized the use of motion capture data by showing how to break it up into small clips and blend clips together to produce desired effects at the appropriate times. However, it is often impossible to build a database that accounts for all the movements that should be made, particularly in response to external perturbations. To deal with this problem, several researchers introduced brief periods of dynamic simulation in response to collisions and then searched for the nearest recorded clip to the resulting pose in order to resume kinematic playback of behaviors[7, 124]. Heck and Gleicher[46] proposed a method for combining families of movement clips into parametrized descriptions that can be used to create novel movements. Motion graphs provide a convenient way for organizing motion capture

data. Our techniques can be used in a complementary fashion.

A problem with kinematically reproducing movements through blending is that it tends to look bad under certain conditions. Humans are particularly sensitive to problems between contact areas (e.g., foot-skate on the ground). One way to deal with foot-skate is to ensure that the blending happens using the contact point as a frame of reference (as opposed the more commonly used waist segment)[110]. The search problem for finding appropriate clips can also be time-consuming in large graphs, but small graphs may not provide enough flexibility. Machine learning approaches trained offline, such as support vector machines[124] and reinforcement learners[110], can be used to create mappings between conditions and clips to speed up the search during online animation. Using physics-based approaches to follow kinematic data can help to clean-up some of the problems caused by blending.

Constrained space-time optimization, first presented by Witkin[117] is still the most common approach used to generate stable dynamics trajectories for achieving arbitrary movement goals. The chief method employed in constructing physically driven humanoid movements is multi-objective optimization (cf., [28, 70, 89]. Using a prioritized or weighted set of cost functions, optimization approaches select optimal motion graph clips or optimal joint actuations. Unfortunately, because of the complexity, optimization is usually offline and not suitable for interactive work.

Optimization-based approaches involve defining a cost-function for a movement over time. The cost function can incorporate many objectives, such as minimizing torque while remaining close to a reference trajectory, but it can be difficult to balance the different objectives (see, e.g., [70]). Optimization produces remarkable

results but is complex and computationally expensive. This dissertation focuses primarily on real-time techniques with minimum planning. Instead we take advantage of captured human behavior to take care of the planning problem.

More recent methods employ offline optimization to plan controllers instead of trajectories. Ye and Liu[119] describe a method for using offline optimization to find an abstract reference trajectory and a feedback policy for following it. These methods typically use the principal of abstraction by optimizing over some feature such as maintaining Center of Mass (COM) over support or minimizing angular momentum[74]. It is also common to use some sort of abstract low-dimensional model such as a spring-loaded inverted pendulum[74] to approximate the movement of the larger system. We show, in Chapter 6 how our approach can take advantage of these low-dimensional models to easily synthesize movement.

In robotics, as in animation, it is common to use simplified models for generating behaviors and treating the differences between the simple model and the full system as noise[115]. We have built off of work by Raibert[85], who built a piston-driven hopping robot by building a body with a mass sufficiently large that it can ignore the inertia of the leg. It chooses foot placement according to the forces necessary to move the center of mass while in the air and rotates its mass while on the ground to counteract angular momentum. With just a few simple feedback rules, it produces dynamic locomotion. We make direct use of this model as an abstraction for more complicated movements.

Other important abstractions such as the linear inverted pendulum[54], zero-moment point (ZMP)[53], and capture point[81] offer additional insights into where

and when to apply force to the ground to achieve stable humanoid balance. We do not directly address these models. We rather use the Raibert hopper as an example to illustrate how a physics-based approach to controlling a character model makes it straightforward to generate movement trajectories with a low-dimensional model in simulation, map those trajectories on to a high-dimensional character model, and then kinematically and dynamically reproduce the movements described by the simple model. A single example of the hopper serves to illustrate the viability and ease of this process.

Real-time techniques for synthesizing kinematic movement without an explicit low-dimensional model typically involve using the pseudo-inverse Jacobian of an end-effector in generalized coordinates to move closer to a target. This process is well-known and covered in many sources, e.g., [36, 102]. Other approaches avoid the challenge of inverting the Jacobian by instead randomly exploring generalized coordinate space for a state that is closer to the target and then interpolating between states[114].

The physics engine used in this research, ODE, is used extensively for animation and robotics research. The popular Robot Operating System (ROS)[83] incorporates ODE for simulating robotic systems. Integrating simulation into ROS allows robotics researchers to test controllers in simulation and then transition to controlling a physical robot without significant cost. This dissertation shows how the built-in physics engine can be successfully employed for additional purposes. Likewise, the SimSpark[76] library uses ODE for collision detection and constrained dynamics simulation for robot soccer in the simulation league of the RoboCup competition. A

lot of research has gone into learning effective humanoid movements in this platform (e.g., [24, 72]). This research into generating humanoid movement for simulated soccer agents mostly uses learning and optimization to build up complex and coordinated behaviors. Our research is complementary to these approaches, making it easier and more stable to do the frame-by-frame tasks of solving inverse kinematics and inverse dynamics.

Simulated approaches are far less constrained than physical robotic systems and can more closely model physical properties of human bodies. They are also free to fall without fear of breaking expensive equipment. These features make it much easier to take advantage of human data and to experiment with highly dynamic movements. This work builds on research from animation and graphics communities. In particular, we focus on physically based character animation. This body of work seeks to generate natural appearing animation by using physically simulated characters driven by forces and torques instead of using purely kinematic methods. An entirely kinematic humanoid will continue walking uninterrupted, even through the air, while projectiles simply pass through its pixels. Physically simulated characters can respond appropriately to unexpected collisions and changing surface properties (see review in [38]).

Forming a bridge from robots to humans, Vona[112] demonstrated that it is easier to use few rather than more degrees of freedom to control a robot (he also provided a kinematic method for simplifying complex systems). Simultaneously controlling many independent degrees of freedom in an effort to manipulate the end-effector of a jointed arm along a specific path is challenging for a human. However, if the

mechanical system can be placed in a mode where the joints are restricted to the manifold that moves the end-effector along the desired path, then the human only needs to control the temporal advancement through that manifold–a much simpler task.

2.5 Summary

We have discussed literature from numerous different areas related to the research and innovations presented in this document. Using interactive virtual reality allows us to capture movement data for natural tasks under tightly controlled conditions. We then use physics-based tools to analyze and then synthesize movement in simulated model. As far as we know, no other work has explored using a physics engine as a single tool for accomplishing the many different tasks addressed by this work; so we have discussed innovations and approaches for handling physics simulation and then separate research for fitting a model to marker data and computing inverse dynamics for a model. An enormous body of techniques exist for controlling and creating movement in robots and animated characters.

We have only been able to scratch the surface in all of these areas. This dissertation takes inspiration from this work and builds on related techniques to create a unified method that is fast, free, and easy to use. The techniques presented in the remainder of this document complement this related work and can help make some activities much easier for programmers, animators, and perhaps, for clinical work and research studying human movement.

Chapter 3

Virtual Reality

Casting intelligence as an embodied phenomenon adds constraints and affordances that can simplify the computations necessary for accomplishing intelligent behavior. Recent technological breakthroughs allow us to use virtual reality to acquire and process large amounts of human movement data under controlled conditions. These data, properly analyzed, can provide insights into how the brain takes advantage of the body's natural dynamics to accomplish everyday tasks.

Motion capture and virtual reality play a key role in recording the human movement during execution of specific tasks under scientifically controlled conditions. We can view each execution of a task as a sample from a probabilistic distribution conditioned on the task parameters and the state of the brain and body. Computational models fit to these data attempt to produce appropriate movements in physical simulation under novel test conditions. This research has significant application to generating realistic character animation and promises to eventually be applied to robotics and clinical scenarios. The first step to achieving these purposes is developing tools capable of gathering the necessary data. This chapter presents our work in developing an interactive virtual environment for studying human behavior. Although virtual reality is an old idea, our implementation represents a novel software engi-

neering effort that includes useful new methods for visualizing data and interesting 3d human-machine interface elements.

A low-latency virtual reality environment supports gathering task-specific movement data under controlled conditions. Producing a usable virtual environment requires incorporating multiple cues that support depth perception in the head-mounted display without adding excessive latency. An interactive virtual environment combines multiple hardware and software systems into a cohesive chunk. Our project incorporates many different sensors for gathering data during a task and combines them with physical simulation and real-time graphics and audio. Sensor data and simulation results influence the virtual environment in real-time. Programmatic scripts define the behavior of the environment in response to significant events to produce controlled experiments. The software produced as part of this research represents a useful artifact for controlled, psychophysical experiments as well as for subsequent analysis and synthesis of movement.

The virtual reality environment consists of several different modules handling different tasks. Figure 3.1 provides an abstract view of the project architecture. The overall design has three large divisions: input, output, and logic.

3.1 Input

We incorporate data from multiple sources. Many resources for producing the environment, such as images and textures for visual detail and sound files for audio feedback, are loaded and organized for rapid access when the program begins. Other input sources provide data continuously. The program core interacts with specific

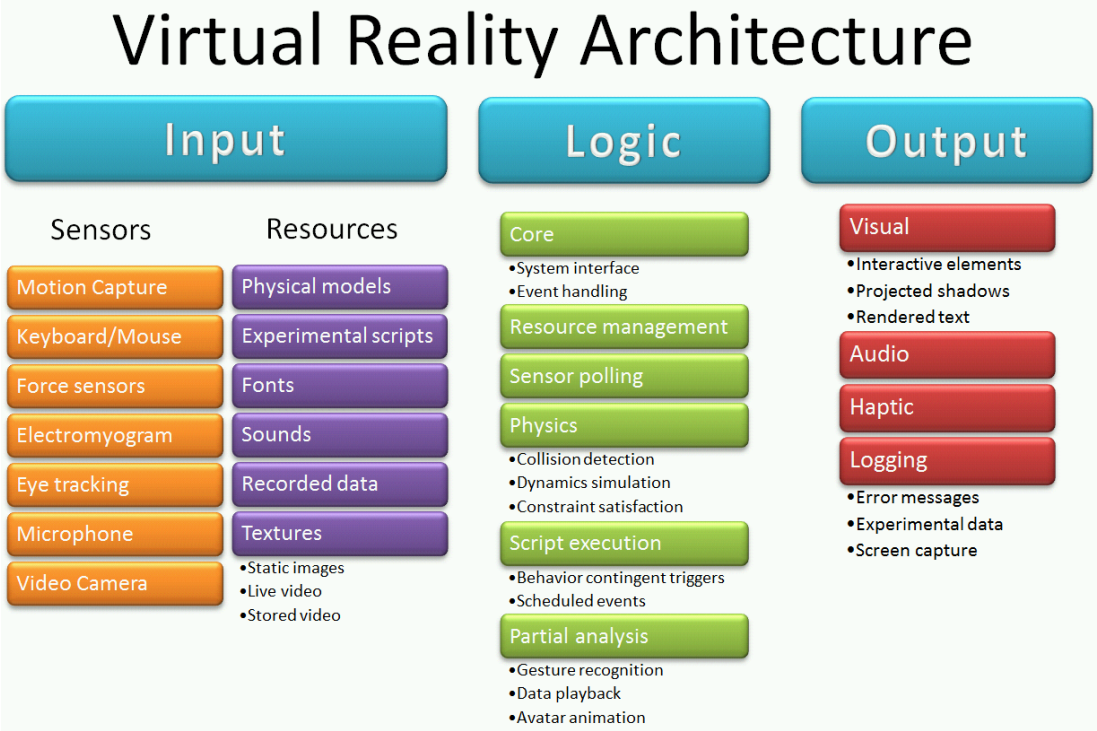


Figure 3.1: Multiple code modules handle the various responsibilities necessary to produce a compelling, interactive, virtual environment and record appropriate experimental data.

libraries to initialize and synchronize the many different sensors and devices used. Some specific devices are covered here:

3.1.1 Motion Capture

Real-time motion capture data allow a human participant to move his or her body to interact with the virtual environment. For motion data, we use a 16 camera Phasespace Impulse motion capture system[1]. This system uses powered LED markers actively pulsing at a high frequency. The cameras identify each marker by its pulse-pattern and triangulate its position within the capture volume. A central server aggregates and processes the camera data. A library provided with the capture system communicates with the server over a local network to receive marker position data in real-time. The server collects data at 480 Hz or less. Each report from the server provides a full frame of data. That is, all of the marker locations at a particular point in time, along with an estimate of the accuracy of each marker's position. The positions are reported in millimeters relative to a reference frame established during calibration of the capture system. our experience has shown that, when calibrated well, the system tracks stationary markers with sub-millimeter precision.

Along with triangulating individual marker positions, the Phasespace server can also take three or more markers rigidly attached to an object and use their positions to compute the object's position and orientation. A Kalman filter smooths these computations slightly. Our module handling motion capture data interacts with the capture system library, directing it to treat the markers attached to the head-mounted

[1] `http://www.phasespace.com/`

display as a rigid body. It then stores the data, transforming the positions into the virtual world's frame of reference. This module allows other program components to access the most recent position of each marker and the position and orientation of any rigid bodies (such as the head). It also performs some minor filtering and provides a finite-differences approximation of the marker velocities.

3.1.2 Electromyogram

When it is useful to record or respond to muscle data, we use a 32-channel Myopac Wireless electromyogram (EMG) system[2]. With this system, electrode pairs placed on the skin record the voltage potentials that result from ions flowing in and out of contracting muscles. These analog measurements are transmitted wirelessly to a base station. A Measurement Computing USB-1616HS device[3] digitizes and provides access to the data. Our module interacts with the appropriate drivers to capture, and filter the data, making them available to other modules.

Data from the EMG provide information on muscle activity. Incorporating EMG into our experimental environment allows muscle activity to be recorded, visualized, and correlated with other sensor data. Muscle data can also produce real-time side-effects in the virtual environment. For example, an experiment may require subjects to maintain a certain level of muscle activity (i.e., co-contraction) while executing a task.

[2]http://www.konigsberginc.com/
[3]http://www.mccdaq.com/

3.1.3 Wii Peripherals

Some experiments require inertial measurements or force measurements. Consumer grade peripherals designed for Nintendo Wii video game consoles provide inexpensive, robust devices that can be easily integrated into a VR environment. The balance board peripheral provides data comparable in quality to lab-grade, medical-use force plates [18], costs a comparatively trivial amount, and communicates wirelessly using known protocols. The balance boards do not provide information on tangent forces (friction), but can still be used to produce useful ground-force measurements for experimental analysis or as input controlling the virtual environment. For example, in a large environment, we employed the force plate as a virtual skate-board, allowing a participant to traverse the environment by leaning in the direction he or she desired to move.

Related devices, such as the wiimote, have accelerometers and gyros for measuring linear and angular accelerations. Buttons and triggers on the devices provide additional input modalities for human subjects using the VR system. An Open Source project, WiiUse[4], provides an API that initializes and communicates with these Bluetooth devices. We encapsulate this functionality to allow other modules continuous access to the state of the force plates or inertial devices. We recognize button presses and forward them to the program's event-handling mechanism, making it simple to design an experiment that requires the subject to push a button in response to a specific event. Convenient portable devices of this sort are often more

[4]https://github.com/rpavlik/wiiuse

reliable for capturing human subject responses than gesture detection that relies on the motion-capture system because the devices do not suffer from marker occlusion.

In the course of this research, we have made some minor contributions to the WiiUse library. In particular, we added code making it possible to record data from more than one balance board at a time. The original code blocked on a single balance board when polling for data, adding unacceptable delay into the system. We adapted the code to use a non-blocking check to see if new balance board data are available, allowing the library to continue processing other data if nothing new has happened. This modification made it possible for us to measure ground force magnitude and location from two feet at once or lay out multiple boards like stepping stones for measuring gait.

3.1.4 Other Inputs

Other sensors are incorporated as needed for different experiments. Although head-tracking gives some sense of the subject's focus of attention. Eye-tracking sensors are useful for pin-pointing where a subject is looking within the environment (Figure 3.2).

We have also incorporated real-time video and audio from external cameras and microphones. Capturing real-time video depends on functionality provided by the DirectShow library. Each frame of video pulled from the camera is written into a buffer and converted into a graphical texture that is available for immediate display

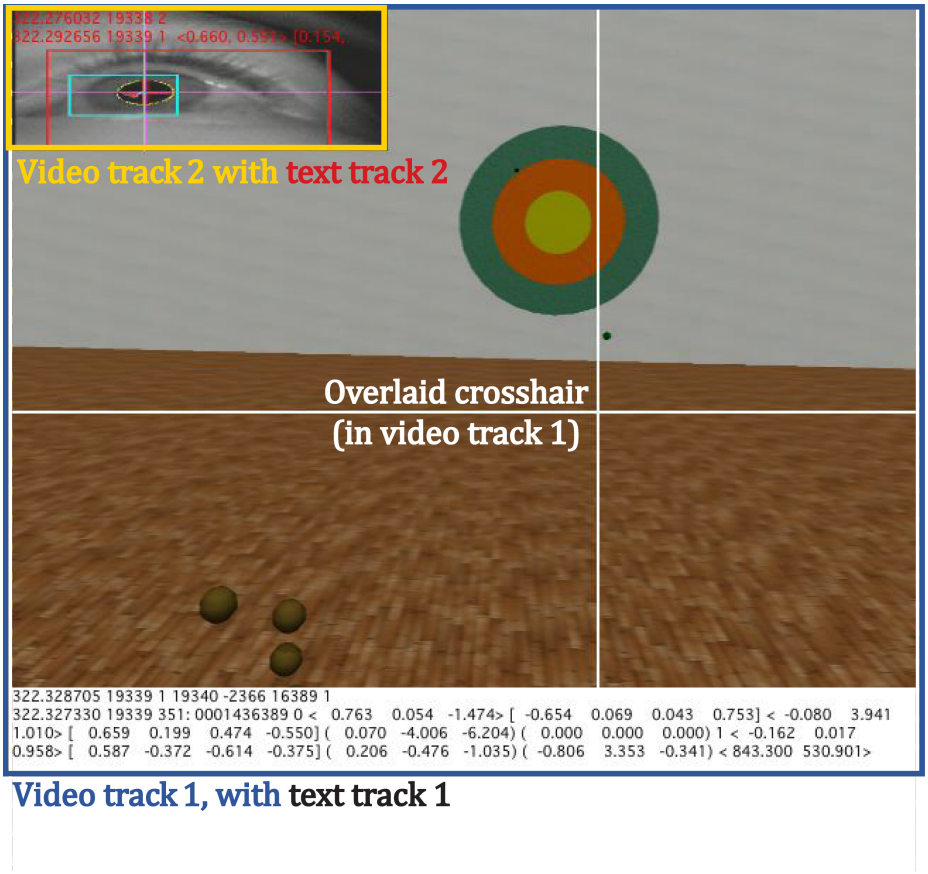


Figure 3.2: A virtual racquetball court with overlaid data from an eye-tracker attached to the head-mounted display[31].

in the virtual environment. We used OpenAL[5] to interact with audio devices and capture microphone data. The library writes audio data samples into small buffers that can be written to file or processed as needed.

We have not made extensive use of microphones in our own experiments. There are, however, many interesting potential uses for captured audio. For example, it would be very interesting to study the correlation of utterances with motor behaviors. For example, a subject could be instructed to walk through and describe a scene. Time-synchronized movement data, eye-tracker data, and spoken data could provide insight into how visual information drives behavior and how behavior adapts based on attention and task. Real-time signal processing could also allow a subject to vocally interact with the virtual environment. Combining microphone data with head-tracking would allow a participant to shout in a particular direction to push objects in the simulated world.

3.2 Output

The responsibility for producing appropriate sound and imagery for the subject lies with the various output modules. These modules interact with system drivers and libraries to generate the audiovisual stimuli necessary for creating an immersive environment. The largest and most complicated of these is the graphics module, which produces real-time video output. Other modules handle audio output, haptic feedback, and recording experimental data.

[5]http://connect.creativelabs.com/openal/

3.2.1 Graphics

We present rendered 3d imagery within a head-mounted display for immersive VR. Our current system uses an nVisor SX111 display[6]. Although the weight of this display becomes uncomfortable after extended use, it provides a high resolution of 1280×1024 pixels to each eye with 102° of horizontal and 64° of vertical field-of-view. The wide field-of-view provides peripheral information that can be very useful in psychophysical experiments.

Multiple issues arise when producing stereo imagery for a head-mounted display. A phenomenon known as simulator sickness[60] is largely inevitable but can be reduced with careful attention to detail. Slight delays and mis-alignments mean that visual data may not exactly match head-movement. Most current head-mounted displays require the eyes to focus on a screen at a fixed distance from the head. These and other issues that are difficult to fully eliminate with current technology lead to conflicting information that contributes to simulator sickness[60].

Depth cues are particularly important for virtual reality. Our first efforts to build an experimental system used a black world with a checkerboard floor and simple colored geometric objects such as cubes and spheres. Touching an object inside the virtual space proved to be very difficult. Although the head-mounted display provided stereopsis and motion parallax, there was not enough information available to accurately localize objects in the virtual world. The brain combines information from multiple sources to understand the spatial configuration of the external world[66]. To

[6]http://www.nvisinc.com/

support scene understanding, we integrated several additional depth cues: reference objects, detailed textures, and projected shadows[48].

Geometric objects such as spheres and cubes can be any size; consequently, seeing and recognizing an object provides little information about how far away the object it. Familiar objects introduced into the space support visual comparison. An object's size can be judged in comparison to an object of known size, providing information about depth based on perspective and foreshortening. Detailed textures applied to objects in the world break up otherwise flat surfaces, making it easier to correctly match stereo correspondences. Textures also look more detailed from up-close than from far away. Semi-realistic projected shadows[48] provided strong visual information communicating 3d structure. It is much easier to judge when your finger will touch an object if your finger projects a shadow onto the object because the shadow and the finger-tip meet at the contact point. Although we have not done any formal validation of the influence of these contributions, we can confidently say that people could not interact with the virtual world before additional depth cues were introduced. These various graphical cues made the world much more immersive, making it possible to actually use the virtual space.

3.2.2 Other Outputs

Auditory cues complement visual cues, providing a compelling experience within the virtual environment. Auditory cues are particularly useful for communicating impact events that occur outside of the field of view. Sound clips served primarily to give auditory cues indicating collisions. Experience using this system for

an experiment involving ball-interception showed that audio was a more compelling collision cue than haptic vibration.

File output is naturally important because the project is used to capture and analyze human data. It is therefore necessary to synchronize and record the data for analysis. Because the virtual environment doubles as a tool for partial analysis, a full trace of data can be written to file and then replayed later, showing the world as experienced by the subject or from an external perspective. The ability to replay data can be a valuable tool for analysis, allowing a researcher to enter the virtual world and observe the subject's captured behavior within that environment after an experiment has ended.

3.3 Logic

Logical modules manage the flow of data and define interactions between inputs and outputs. Our virtual environment uses looping iterations to coordinate the sensor inputs with the audio-visual data presented to the subject. The program core takes data and events coming from one source and uses them to trigger appropriate responses in other modules. The core also handles the responsibility of properly initializing all other modules and integrating them as necessary for a particular experiment. Other logical modules work under the direction of the core to process and transform data internally before they are passed to output modules.

3.3.1 Core

The core module organizes all other modules and interfaces with the computer operating system. To handle windowing (opening a graphical interface) we use the Simple DirectMedia Layer (SDL) libraries[7]. These libraries provide cross-platform compatibility and make it relatively painless to connect some of the output modules, such as the display and audio, to the appropriate hardware for them to function correctly. These also provide basic event loops to capture keyboard and mouse input.

The primary purpose of the core module is to initialize, update, and terminate all other modules. During program execution, the core module follows a set of rules to distribute information between modules and activate modules as needed. The processing order of these rules is important for maintaining up-to-date data and minimizing latency.

3.3.2 Physics

We use the Open Dynamics Engine (ODE) to simulate constrained physical dynamics. After updating the sensor input channels, the core logic converts relevant data to physical quantities such as forces and velocities for use in the simulation engine. For example the latest motion capture marker positions inform their counterparts in the simulation so that the simulated markers go to the correct place. We discuss the physical simulation at length in Chapter 4.

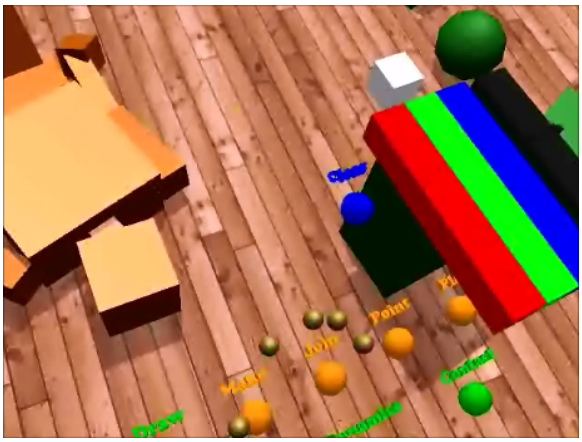


Figure 3.3: Virtual buttons, labeled with 3d text, make it easy to manipulate system properties without leaving the virtual world. Small golden spheres near the bottom of the figure follow motion capture markers on the hand, making it possible to detect collisions between the fingers and virtual interface elements.

3.3.3 Internal processing

After the physics engine simulates dynamics forward in time, interesting events such as a collisions can be reported as events to other modules and the new state of the virtual world is available for the display module. In this way, the physics engine makes interaction possible between virtual objects and a human subject. When a subject touches a virtual object, the physics module detects a collision and reports it as an event. The core module can then pass this information to scripted logic that may create a contact constraint forcing the object away from the subject.

Alternatively, the core logic may treat the object as a "virtual button" that triggers a sound or turns off gravity (Figure 3.3). Other user interface elements are easily implemented. Detecting where motion capture markers on the hand intersect

[7] `http://www.libsdl.org/`

with virtual interface elements makes it simple to control continuous variables. For example, the red-green-blue "sliders" on the right side of Figure 3.3 comprise a "color-picker'. Touching the boxes at different places changes the amount of each component in the "active color" used with other elements in the virtual world.

Using 3d graphical user interface elements in this way proves very useful when designing or testing an experiment. Rather than go back and forth between the motion capture space and a desktop keyboard, a researcher can employ a "virtual console" that manipulates important variables while pilot testing an experimental idea. In a way, the virtual console allows you to program within the virtual environment, making the design and test process considerably easier. We are not aware of virtual reality literature exploiting physical simulation to create these types of virtual buttons and sliders as human-interface elements for controlling the environment.

3.4 Capturing Behavior

Psychophysical experiments reveal computational principles underlying behavior. The virtual environment currently serves as a tool for eliciting and capturing task-specific behaviors to better understand low-level functions of the brain in a natural task and environment[32, 33].

Using our virtual racquetball court and humanoid model, we can systematically produce controlled ball trajectories and observe how a human accomplishes the goal of hitting the ball to a particular location[32]. We also record locomotion sequences and other relevant movements. Recording data to file produces large sequences of marker trajectories through 3D space, possibly associated with the envi-

Figure 3.4: An example virtual reality environment used for experimentation.

ronmental conditions under which they were recorded. Visualizing and analyzing the data can provide some insights into the movement features underlying the generation of the task-specific movement.

Figure 3.5 illustrates a task in which a human attempted to swing, with the right hand, at a ball launched repeatedly along an identical trajectory. Although the ball path was constant, the movement used to swing at it demonstrated significant variance, providing some insight into the manifold of acceptable swinging movements for that particular set of conditions. The data also suggested that for that particular task, the human adopted a strategy that involved moving the hand along a parabolic path indicating prediction of the ball's future path. By so doing, the human could increase the probability of hitting the ball because it did not matter when the hand

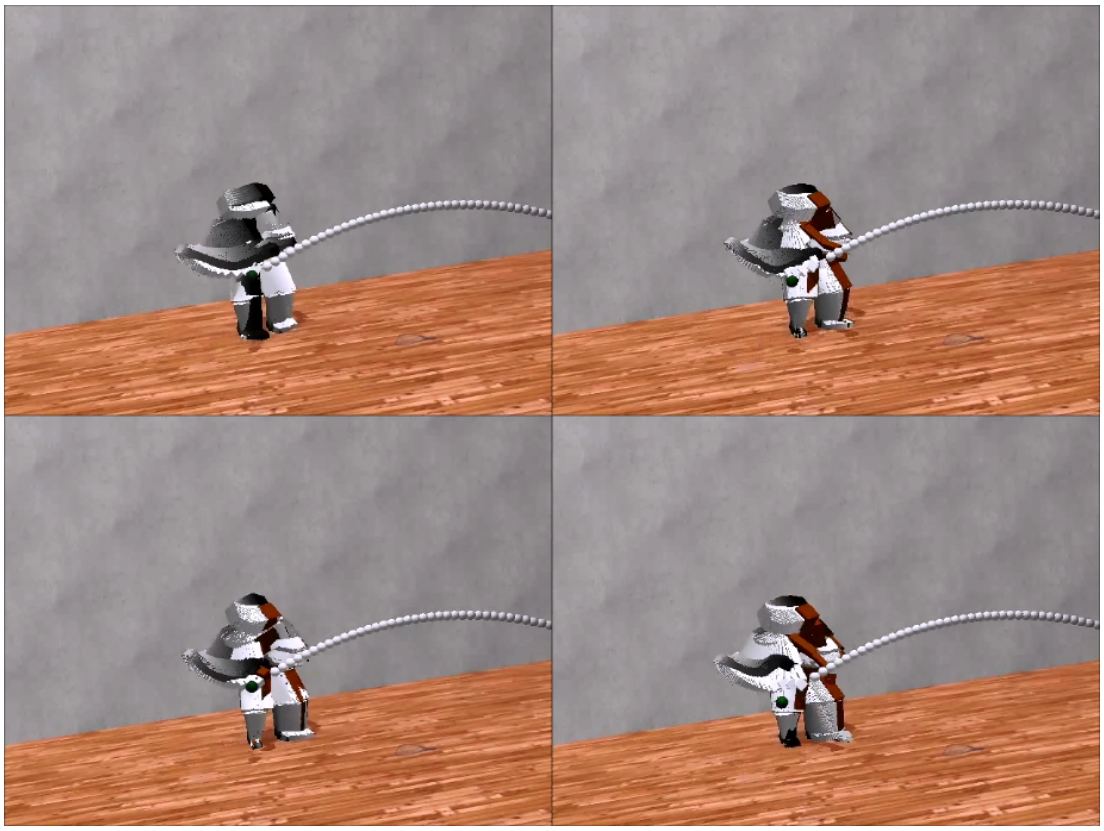


Figure 3.5: Novel visualization of swinging to hit a ball in virtual reality. Grayscale values show the human data and ball over time. Lighter values are more recent. The final trajectory of the human hand suggests that it is following the predicted parabolic curve of the ball, increasing the chances of intersection.

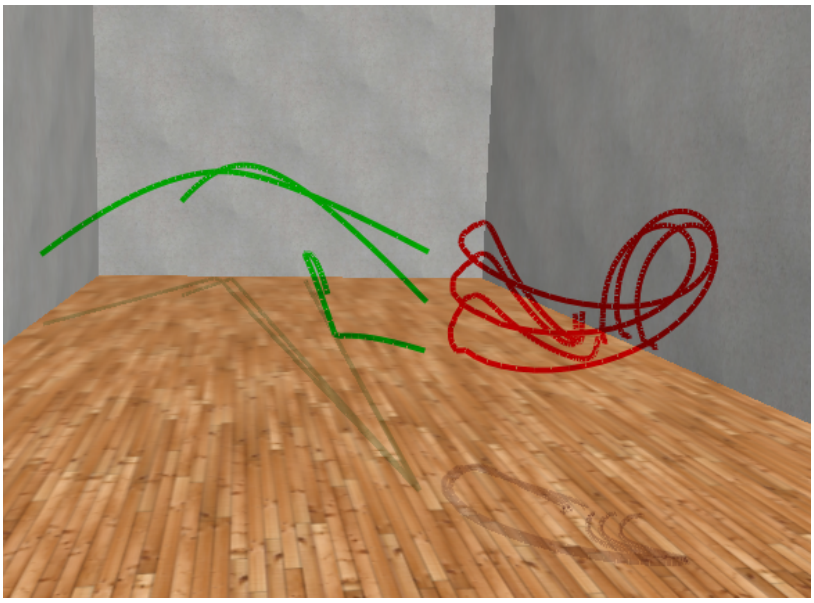


Figure 3.6: Green lines show ball paths resulting from swings at 1m, 1.5m, and 2m height. Red lines show the mean trajectory taken by the center of the racquet for back-hand swings at balls of the different heights.

hit the ball.

Figure 3.6 compares the mean path of swings under three different conditions: an unmoving ball placed 1m, 1.5m, or 2m above the ground. The first and last parts of the trajectories are very similar, but the middle part differs drastically, suggesting three phases to this type of swing. The first and last phases are possibly based on a body-relative feature, while the middle phase must use the ball location as a reference frame in order to properly connect and produce the desired outcome.

Figures 3.5 and 3.6 are just a few examples of the power of virtual reality for capturing and analyzing movement. Plotting the experimental data within the virtual environment used to capture it provides incredibly useful context information. These novel visualizations and the ability to explore the data in 3d communicates information that might not otherwise be available.

3.5 Summary

Virtual reality is a very useful tool for behavioral experiments. Building such a platform represents a non-trivial amount of software engineering, integrating multiple independent hardware and software elements into a cohesive, real-time system. Simulated physics plays an important role in the virtual environment, computing forward dynamics for the simulated world as well as making it simple to interact with virtual interface elements. We will see in future chapters how the same physics simulation library also facilitates analysis of the movement.

There are many interesting avenues open additional research for the virtual

environment. The possibilities for different psychophysical experiments are endless. There are also numerous interesting paths for artistic application of virtual reality. One might produce visual representations of audio recorded from microphones and leave them positioned within the virtual environment to be played again when a participant returns to the point. The audio could generate a particle effect that cascades through the virtual environment, colliding and interacting with other elements. It may be possible to simulate appropriate audio based on material and collision properties for more compelling VR (see [120]). Virtual and augmented reality technologies are growing at a rapid pace, emphasizing the importance of tools such as those we have developed and making the future in this area very exciting.

Chapter 4

Physical Simulation

The principal contribution of this dissertation is a method for taking advantage of robust numerical simulation code to accomplish several important tasks related to analyzing and synthesizing humanoid movement. This chapter covers the mathematics underlying the rigid body simulation software used in this work with a focus on the details that make it possible to adapt the numerical simulation for other purposes. The bulk of this chapter presents equations already incorporated into simulation libraries before we began working on it. Although our specific derivation explaining the equations may be somewhat novel, most of the work implementing the fast and robust rigid-body simulation provided by ODE was done by others.

The principal insight in this chapter is that ODE can be used as an effective controller. We present a derivation of the mathematics underlying the physics simulation. The derivation comes from directly analyzing the ODE codebase and it consequently differs from other derivations using Lagrange multipliers to arrive at the same final result (e.g., [36]). We present another derivation illustrating the equivalence between softened constraints in ODE and implicit springs. Once again, the derivation is something we have worked through ourselves and differs from other methods for arriving at the same conclusion (e.g., [17]). These explanations serve to

show how our particular innovations fit into the overall system and how the system makes it possible to use simulation as a multi-purpose tool for movement analysis. Because constraints in ODE are mathematically equivalent to implicitly stable damped springs (a known fact) and damped springs are mathematically equivalent to proportional-derivative (PD) controllers (another known fact), we arrive at the obvious but currently under-appreciated conclusion that ODE can be used as a powerful tool for controlling humanoid models and that this control approach can be used to analyze and synthesize humanoid movement in a single robust framework that integrates seamlessly with software systems already using ODE for forward dynamics simulation.

Our efforts to explain the mathematical and programmatic details that make it possible to use ODE as a stable controller also led to several improvements of the public codebase. Some of these contributions take the form of bug fixes submitted to correct simple errors that have persisted in the project. For example, we repaired some basic problems with matrix inversion code and collision detection code. Other contributions added flexibility or stability to existing constraints. We fixed logical errors in certain angular constraints. We incorporated optional rolling and spinning friction computations into contact constraints that occur between two colliding surfaces. These friction computations can be used to improve interactions between a character's feet and the ground. We also implemented an implicit computation of gyroscopic torques to improve the stable simulation of rotating bodies. We created new generalized joint-types that, for example, allow you to constrain the foot to move toward a certain height without constraining its horizontal position. We only

mention these contributions briefly here. The details of their implementation have been submitted and are freely available through the ODE public repository[99]. This chapter does, however, cover a particular insight gained from studying the constraint mechanism that allows us to significantly increase the types of constraints that can be created in ODE. This insight recognizes that constraints in ODE can be represented, generally, as piecewise linear manifolds through force-acceleration space with up to three linear segments, but the limitation to three segments is unnecessary. Adding additional linear segments to a constraint manifold results in constraints (and consequently controllers) with powerful properties. These properties are discussed at the end of the chapter.

4.1 Constrained Simulation

Constrained physical simulation involves mathematically modeling multiple physical entities with physical properties that determine the entities' temporal evolution. These entities, each called a *rigid body*, have dynamic state such as position, velocity, orientation, and angular velocity. They also have properties that are generally constant over time, such as mass and volume. The dynamic state of each body changes according to equations of motion and constraints. The equations of motion capture physical laws such as conservation of energy and momentum while constraints describe relationships between the bodies.

Systems of multiple rigid bodies constrained to move together in specific ways, such as through hinge or ball-and-socket joints, are known as articulated models. The dynamic state of an articulated model evolves according to differential equa-

tions that can be difficult or impossible to solve analytically as a function of time. Simulation software libraries, such as ODE[99], typically use numerical integration to approximate the solution. Discrete physical simulation numerically approximates the solution to the equations of motion of an articulated body by iterating over time in small steps. Each step consists of multiple phases: collision detection, constraint construction, constraint solving, and then integration. These steps are described in detail below.

4.2 Collision Detection

Rigid bodies are represented as logical entities with collision geometry that maintains a constant position and orientation relative to the body's frame of reference. A collision occurs when the geometries of two bodies touch or overlap. Collisions between rigid bodies represent sharp discontinuities in the differential equations describing their state. Because physics simulation occurs over discrete time steps, collisions occur with varying degrees of overlap. Collision detection routines find bodies that are touching or overlapping so that the temporal dynamics can be constrained to prevent further interpenetration.

The position and orientation of bodies update in discrete steps. If the relative velocity of two bodies over a single timestep is greater than the magnitude of their dimensions, they can pass through each other without ever registering a collision. This phenomenon is known as tunneling. This work does not focus on collisions, but there are different methods for dealing with this possibility. Moving collision geometries can be stretched out in the direction of their velocities so that all possible

collisions will be found. The easiest thing to do is ensure that the size of the discrete time step is sufficiently small relative to the fastest moving bodies so that tunneling is not a problem. However, if a specific non-penetration condition is particularly important, it can be added as an inequality constraint bounding the relative velocity of two points so that the bodies cannot move fast enough to interpenetrate. This method is not scalable because it involves adding a velocity constraint between every potentially colliding pair of bodies, but it could be used to enforce a small number of safety constraints to keep a system from colliding with people or other equipment. The work in this document always employs the collision behavior of allowing bodies to penetrate slightly and then applying constraints that prevent further penetration and force the bodies to move apart.

Collision detection happens in two phases: broad and narrow. The purpose of broad-phase collision detection is to organize the geometry of the different bodies into data structures that cull most potential collisions thereby avoiding the worst case of $\mathcal{O}(n^2)$ comparisons. These methods involve sorting objects into bounding volumes such as axis-aligned bounding boxes. These bounding volumes, organized hierarchically, facilitate rapid comparison between multiple geometries. If outer volumes do not intersect, then there is no need for running the computationally intensive narrow-phase collision tests.

In narrow-phase collision detection, specialized algorithms test pairs of bodies for overlapping. The collision tests typically involve using closed-form equations of the geometry to analytically determine the intersection of two surfaces. For some geometries, analytic equations for computing intersection are intractable or unknown.

When closed-form collision computation is not possible, there are general-purpose algorithms for finding approximate collision information[39]. When surfaces intersect, the collision algorithms compute penetration depth and produce a set of collision points and direction vectors that can be used to prevent the bodies from moving closer together.

This dissertation makes very little use of collision detection. The virtual reality environment processes collision between different objects, a key element in creating realistic tasks in natural environments. When modeling human movements, however, we assume that the human body does not collide significantly with itself and so typically only process collisions between the model and the ground. Analyzing movements where this assumption is violated, such as when one limb rests on another (crossing one's legs while sitting) would necessitate internal collisions. These types of movements and poses are not addressed in this dissertation. Collisions between the model and the ground, however, play an important role in analyzing and synthesizing movement data such as walking.

4.3 Dynamics

Once collisions are found, the programmer needs to decide how the collisions should affect the simulation. Generally, collision handling involves creating a constraint between the colliding bodies. We will return to the subject of creating constraints later. At this point we will discuss the simulation itself after introducing a table of necessary symbols.

4.3.1 Notation

Physical simulation involves an annoyingly large number of different variables to represent constraints and relevant physical quantities. Scalars are represented with lower-case, un-bolded symbols: x. Bold lower-case symbols represent column vectors,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We use bold, upper-case symbols to represent matrices:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$$

We use dot-notation to indicate time derivative: $\dot{x} = \frac{dx}{dt}$. The circumflex accent indicates a 3d vector being used as a skew-symmetric matrix representing a cross-product operation:

$$\hat{\boldsymbol{x}}\boldsymbol{y} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \boldsymbol{x} \times \boldsymbol{y}$$

Coordinates are typically relative to a "global reference frame" established arbitrarily and used for consistency. Discussing physics requires understanding multiple frames of reference and their relationships. Full treatment of this subject is beyond the scope of this document. Additional information is available in [31]. A tilde, $\tilde{\boldsymbol{x}}$, indicates a quantity that uses a local reference frame, e.g., a body-relative frame, rather than the global frame. We abuse subscripts to indicate that a quantity refers to a specific dimension, a particular rigid body, a point in time, and in other cases that warrant extra information to distinguish related symbols. We attempt to clarify the subscripts when necessary to remove ambiguity. We attempt to introduce

Symbol	Meaning
$\boldsymbol{x}$	position or state of one or more rigid bodies
$\dot{\boldsymbol{x}}$	velocity (usu. linear and angular)
$\ddot{\boldsymbol{x}}$	acceleration (usu. linear and angular)
$\boldsymbol{R}$	rotation matrix representing orientation of a body
$\boldsymbol{\omega}$	angular velocity
$\boldsymbol{q}$	quaternion representation of an orientation or rotation
m	mass of a single rigid body
$\boldsymbol{M}$	mass matrix
$\boldsymbol{I}$	identity matrix
$\boldsymbol{\mathfrak{I}}$	moment of inertia tensor
n_b, n_c	number of bodies, number of constraints
$\boldsymbol{\alpha}, \boldsymbol{\beta}$	stabilizing parameters added to the equations of motion
$\phi()$	error or energy function for a single constraint
$\boldsymbol{J}$	matrix of partial derivatives of constraint error functions
h	timestep
$\boldsymbol{f}$	forces (and torques)
$\boldsymbol{\tau}$	torques
$\boldsymbol{\lambda}$	constraint forces

Table 4.1: Meanings of specific symbols used to discuss dynamic simulation

new symbols within the text, however, Table 4.1 presents specific symbols and their meanings for reference.

For conciseness in notation, we will typically combine angular and linear quantities as a single symbol. This representation is used loosely for position and orientation because orientation does not conveniently fit into a 3×1 vector. Fortunately, angular velocity and angular acceleration do combine well with linear velocity and acceleration, and it is these quantities, $\dot{\boldsymbol{x}}$ and $\ddot{\boldsymbol{x}}$ that feature primarily when dealing with a constrained system. We will also represent the state of multiple bodies using a single symbol when convenient. For example, for a system with two bodies, we will

represent the combined linear and angular accelerations (a 12d vector) as $\ddot{\boldsymbol{x}}_t$. For this same 2-body system, Newton's law relating force, mass, and acceleration is as follows:

$$\begin{bmatrix} \boldsymbol{f}_{1t} \\ \boldsymbol{\tau}_{1t} \\ \boldsymbol{f}_{2t} \\ \boldsymbol{\tau}_{2t} \end{bmatrix} = \begin{bmatrix} m_1\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\mathcal{I}}_{1t} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & m_2\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\mathcal{I}}_{1t} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{x}}_{1t} \\ \dot{\boldsymbol{\omega}}_{1t} \\ \ddot{\boldsymbol{x}}_{2t} \\ \dot{\boldsymbol{\omega}}_{1t} \end{bmatrix} \Rightarrow \boldsymbol{f}_t = \boldsymbol{M}_t\ddot{\boldsymbol{x}}_t$$

where $\boldsymbol{I}$ and $\boldsymbol{\mathcal{I}}_{it}$ are 3×3 block matrices.

4.3.2 Dynamic State

Coordinates in the simulation world are defined relative to an arbitrary origin and basis set of directions. We refer to this inertial frame as the "global frame". Each rigid body also has its own point of reference and set of directions. Any point in the global frame can also be described relative to a body's frame of reference. It is convenient to define the point of reference of a body as its center of mass and use its principal inertial axes of symmetry as directions.

The position of the center of mass and orientation of a body within the global frame are here defined as $\boldsymbol{x}$ and $\boldsymbol{R}$ respectively. In 3d space, $\boldsymbol{x}$ is a 3×1 vector:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where x, y, and z are the distance from the origin along each of the three directions that establish the global frame of reference. For consistency, we deal with these distances in meters and assign "up" to the positive z axis. The orientation $\boldsymbol{R}$ of a body is a 3×3 orthonormal matrix whose columns give the body's local direction

frame relative to the global frame:

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Conservation of momentum makes it necessary to keep track of the time derivative of these quantities: $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{R}}$. Instead of explicitly representing $\dot{\boldsymbol{R}}$, it is convenient to keep track of the angular velocity:

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

The relationship between these quantities is

$$\dot{\boldsymbol{R}} = \hat{\boldsymbol{\omega}}\boldsymbol{R}$$

Representing orientation as a 3×3 matrix can be unwieldy. To properly represent an orientation (or pure rotation) it must be an orthonormal matrix. An orthonormal matrix uses nine elements to represent a property with only three degrees of freedom. Unfortunately, any three-element representation of orientation suffers from singularities[42]. We make use of unit-length quaternions to represent orientations and changes in orientation. Quaternions are convenient because of their close relationship to angular velocities. Quaternions are similar to an axis-angle representation of a rotation. A quaternion $\boldsymbol{q}$ represents a rotation by θ around unit vector $\boldsymbol{v}$ with four elements:

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} \\ v_x \sin\frac{\theta}{2} \\ v_y \sin\frac{\theta}{2} \\ v_z \sin\frac{\theta}{2} \end{bmatrix}$$

From an arbitrary angular velocity $\boldsymbol{\omega}$, we can make a quaternion that represents the change in rotation that would occur during a timestep of h. After finding the amount of rotation $\theta_t = h\|\boldsymbol{\omega}_t\|$, one might naively find a "rotation quaternion" by normalizing $\boldsymbol{\omega_t}$ and then re-scaling by $\sin\frac{\theta_t}{2}$ for a final quaternion: $\boldsymbol{q}_t = \begin{bmatrix} \cos\frac{\theta_t}{2} \\ \frac{\boldsymbol{\omega_t}}{\theta_t}\sin\frac{\theta_t}{2} \end{bmatrix}$. However, the normalization step becomes unstable as θ_t approaches zero. To avoid that instability, we use the "sinc" function where $\operatorname{sinc}\theta = \frac{\sin\theta}{\theta}$. The sinc function allows us to remove the discontinuity that would result from division by zero and adds numerical stability. When θ is small, $\operatorname{sinc}(\theta)$ can be approximated to within machine precision using the first two non-zero terms of its Taylor expansion (see [42]). The result is a discrete-time "rotation quaternion":

$$\boldsymbol{q}_t = \begin{bmatrix} \cos\frac{\theta_t}{2} \\ \frac{h}{2}\operatorname{sinc}\frac{\theta_t}{2}\boldsymbol{\omega_t} \end{bmatrix} \tag{4.1}$$

Given n_b bodies, the dynamic state of the i^{th} body at time t is its position, orientation, linear velocity, and angular velocity: $\left\{ \begin{array}{cccc} \boldsymbol{x}_{it} & \boldsymbol{R}_{it} & \dot{\boldsymbol{x}}_{it} & \boldsymbol{\omega}_{it} \end{array} \right\}$. We will assume that all of these values are framed in the global coordinate system unless specified otherwise. The body dynamics are also affected by the body's constant mass m_i and inertia tensor $\boldsymbol{\mathcal{I}}_{it}$. The moment of inertia tensor, $\boldsymbol{\mathcal{I}}$, is indexed by time because the body's orientation changes how the the mass is distributed relative to the world frame: $\boldsymbol{\mathcal{I}}_{it} = \boldsymbol{R}_{it}\tilde{\boldsymbol{\mathcal{I}}}_i\boldsymbol{R}_{it}^{\mathrm{T}}$. We assume that the inertia tensor is constant relative to the body-local frame of reference (i.e., bodies are rigid).

In simulation, the forces $\boldsymbol{f}$ applied to the rigid bodies come from three general sources. These are constraint forces ($\boldsymbol{f_c}$), gravitational and gyroscopic forces ($\boldsymbol{f_g}$), and user/control forces ($\boldsymbol{f_u}$): $\boldsymbol{f} = \boldsymbol{f_c} + \boldsymbol{f_g} + \boldsymbol{f_u}$.

4.3.3 Integration Step

When a force is applied to a body, it translates into acceleration that is inversely proportional to the mass. Velocity is the time integral of acceleration, $\dot{\boldsymbol{x}}_t = \dot{\boldsymbol{x}}_{i0} + \int_0^t \boldsymbol{M}_i^{-1} \boldsymbol{f}_t dt$, and position is the time integral of velocity, $\boldsymbol{x}_{it} = \boldsymbol{x}_{i0} + \int_0^t \dot{\boldsymbol{x}}_{it} dt$. Because $\boldsymbol{f}_t$ may depend on $\boldsymbol{x}_t$ and $\dot{\boldsymbol{x}}_t$ as well as on discontinuous collisions and control inputs, analytic descriptions of body state are not usually possible. Instead we discretize the equations of motion and use a small, discrete timestep, h, to numerically approximate system dynamics. The most obvious thing to do is to linearize the force function, $\boldsymbol{f}_t$, and then take all the quantities from time t and use them to find the state at time $t + h$:

$$\dot{\boldsymbol{x}}_{t+h} = \dot{\boldsymbol{x}}_t + h\boldsymbol{M}^{-1}\boldsymbol{f}_t$$

$$\boldsymbol{x}_{t+h} = \boldsymbol{x}_t + h\dot{\boldsymbol{x}}_t$$

This equation is known as explicit Euler integration[44]. Unfortunately, the delay between the time that forces are applied and the time that the position changes makes this method unstable for most purposes.

Just as simple, but much more stable is "semi-implicit Euler" integration (also called "symplectic Euler" integration). Integrating this way involves using the future velocity for computing position. In this way, a force applied at time t is observed immediately as a change in position at time $t + h$:

$$\dot{\boldsymbol{x}}_{t+h} = \dot{\boldsymbol{x}}_t + h\boldsymbol{M}^{-1}\boldsymbol{f}_t \tag{4.2}$$

$$\boldsymbol{x}_{t+h} = \boldsymbol{x}_t + h\dot{\boldsymbol{x}}_{t+h} \tag{4.3}$$

Although we lump orientation and position together as a single symbol, in practice there are a few distinctions that need mentioning. For example, gravity only applies to the linear state, while gyroscopic torques only apply to angular state. Gravitational forces are very straightforward, $\boldsymbol{f}_{\text{grav}} = \boldsymbol{M}\boldsymbol{g}$, where $\boldsymbol{g}$ indicates the direction and magnitude of gravitational acceleration and is often very simple; e.g., for a single rigid body $\boldsymbol{g} = \begin{bmatrix}0 & 0 & -9.8 & 0 & 0 & 0\end{bmatrix}^{\text{T}}$.

Rotation is a non-linear phenomenon. However, we can approximate the motion of a rotating body by adding torques that imitate gyroscopic effects (for more information, see [15]). Gyroscopic torques are applied to maintain conservation of angular momentum. Explicitly applying gyroscopic torques to bodies allows us to treat the rest of the system as though it conserved angular velocity rather than angular momentum. Thereafter, we can deal with the combined linear and angular quantities as a linear system (since conservation of linear velocity is the same thing as conservation of linear momentum).

The gyroscopic torques for each body are linearly approximated by

$$\boldsymbol{f}_{\text{gyro}} = \begin{bmatrix}\boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \hat{\boldsymbol{\omega}}_t\end{bmatrix} \begin{bmatrix}\boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\mathcal{I}}_t\end{bmatrix} \begin{bmatrix}\boldsymbol{0} \\ \boldsymbol{\omega}_t\end{bmatrix}$$

These forces are zero if the three principal moments of the inertia tensor are equal. Otherwise, they represent the forces necessary for conservation of angular momentum. Unfortunately, this approximation tends to introduce energy into the system. We have done some work to reduce this problem in ODE by adding in additional terms as described in [15], but that work is largely outside of the scope of this dissertation.

The constrained system is solved using mostly accelerations and velocities. At

the end, however, it is necessary to integrate the velocities into new positions and orientations. Position and orientation are updated differently. For position, it is sufficient to multiply the linear velocity by the timestep and add it to the current position. Adding angular velocity to orientation is not as straightforward. We integrate angular velocity into orientation by converting $\boldsymbol{\omega}_{i(t+h)}$ into a quaternion using Eq. 4.1. We then use the quaternion to rotate the current orientation forward in time following [42].

4.4 Constraint Equation

When a rigid body is moving or spinning freely through space, the integration equations are sufficient to simulate dynamics. Adding constraints modifies the bodies' movements. Maintaining a relationship between two bodies requires forming a constraint on the state of the bodies. The integration equations tell us how to go from force to velocity and from there to position and orientation. To simulate an articulated model using maximal coordinates, we need to know what forces constraints apply to the bodies in the system.

In order to find the constraint forces, we must be able to mathematically describe the constraint. We will define a multi-dimensional function over the combined position and orientation of all bodies in the system, $\boldsymbol{\phi}(\boldsymbol{x}_t)$, that produces a vector of size n_c specifying how much each constraint is violated, where n_c is the number of constraints acting on the system. For example, if the i^{th} constraint keeps body b_2 a distance d above body b_1 in the z direction, we would have $\phi_i(\boldsymbol{x}) = x_{2z} - x_{1z} - d$. If b_2 is not separated from b_1 by a distance of d in the z direction, $\phi_i(\boldsymbol{x})$ reports the signed

magnitude of that constraint error. For additional information on forming constraint equations, see [36, 100].

In general, the error for a constraint is non-zero. Given a measure of the error for a given state, we seek to find constraint forces, $\boldsymbol{f_c}$, that reduce the error over subsequent time steps[11]. Specifically, over the timestep h, we seek a force to reduce the magnitude of the constraint error by a fraction α. That is

$$\boldsymbol{\phi}(\boldsymbol{x}_{t+h}) = (I - \boldsymbol{\alpha})\boldsymbol{\phi}(\boldsymbol{x}_t) \tag{4.4}$$

where $\boldsymbol{\alpha}$ is a $k \times k$ diagonal matrix with each $\alpha_i \in [0, 1]$ representing the fraction of error reduction over a time step. In ODE, the α value is controlled through the *error reduction parameter* (ERP) which can be set independently for each constrained degree of freedom. In practice, it is not possible to remove constraint error completely ($\alpha = 1$) when using maximal coordinates because of error introduced by the various approximations employed to make the simulation linear and fast. Values of α typically fall within $[0.2, 0.8]$. Manipulating this value results in useful elastic and damping effects discussed later.

We use the symbol $\boldsymbol{J}_t$ to represent the $n_c \times 6n_b$ matrix of partial derivatives of $\boldsymbol{\phi}(\boldsymbol{x}_t)$. This matrix is a linear approximation of how the constraint error for each of the n_c constraints changes when the positions and orientations of the bodies change:

$$\boldsymbol{J}_t = \nabla\boldsymbol{\phi}(\boldsymbol{x}_t) = \begin{bmatrix} \frac{\partial\phi_1}{\partial x_{1t}} & \cdots & \frac{\partial\phi_1}{\partial x_{(6n_b)t}} \\ \vdots & \ddots & \vdots \\ \frac{\partial\phi_k}{\partial x_{1t}} & \cdots & \frac{\partial\phi_k}{\partial x_{(6n_b)t}} \end{bmatrix}$$

Find the constraint forces that satisfy Eq. 4.4 involves removing all references to unknown future quantities. The Taylor expansion of $\boldsymbol{\phi}(\boldsymbol{x}_{t+h})$ at $\boldsymbol{x}_t$, truncated after

the first order term, approximates the future constraint error:

$$(I - \boldsymbol{\alpha})\boldsymbol{\phi}(\boldsymbol{x}_t) = \boldsymbol{\phi}(\boldsymbol{x}_{t+h}) \approx \boldsymbol{\phi}(\boldsymbol{x}_t) + \boldsymbol{J}_t(\boldsymbol{x}_{t+h} - \boldsymbol{x}_t) \tag{4.5}$$

This truncation has the effect of treating all constraints as linear. Many constraints used to simulate various joints only have linear terms. Others, however, contain higher-order terms and this truncation is one potential source of error in simulation.

Combining the two integrator equations, Eqs. 4.2 and 4.3, gives the future position/orientation in terms of the present position, velocity, and forces:

$$\boldsymbol{x}_{t+h} = \boldsymbol{x}_t + h\dot{\boldsymbol{x}}_t + h^2\boldsymbol{M}_t^{-1}\left(\boldsymbol{f}_{ct} + \boldsymbol{f}_{gt} + \boldsymbol{f}_{ut}\right) \tag{4.6}$$

Equations 4.4, 4.5, and 4.6 combine to substitute away all references to future quantities:

$$(I - \boldsymbol{\alpha})\boldsymbol{\phi}(\boldsymbol{x}_t) = \boldsymbol{\phi}(\boldsymbol{x}_t) + \boldsymbol{J}_t\left(\boldsymbol{x}_t + h\dot{\boldsymbol{x}}_t + h^2\boldsymbol{M}_t^{-1}\left(\boldsymbol{f}_{ct} + \boldsymbol{f}_{gt} + \boldsymbol{f}_{ut}\right) - \boldsymbol{x}_t\right) \tag{4.7}$$

This leaves one unknown vector at time t: the constraint forces $\boldsymbol{f}_{ct}$. Rearranging and simplifying, we get

$$\boldsymbol{J}_t\boldsymbol{M}_t^{-1}\boldsymbol{f}_{ct} = -\frac{1}{h^2}\boldsymbol{\alpha}\boldsymbol{\phi}(\boldsymbol{x}_t) - \frac{1}{h}\boldsymbol{J}_t\dot{\boldsymbol{x}}_t - \boldsymbol{J}_t\boldsymbol{M}_t^{-1}\left(\boldsymbol{f}_{gt} + \boldsymbol{f}_{ut}\right) \tag{4.8}$$

Note that in rearranging the terms this way, we divided both sides by the squared timestep, h^2, effectively changing the problem from one dealing with positions to one dealing with accelerations. This conversion is possible because of the relationship established between acceleration and position by the semi-implicit Euler integrator.

Equation 4.8 is almost the equation that ODE solves when simulating physics. The right hand side is a desired acceleration. The first term on the right is the

acceleration that would result in a velocity that would remove a fraction (α) of the constraint error. The second and third terms account for the effects of momentum (current velocity), gravity, and other forces (e.g., user control forces) applied to the bodies. Each constraint becomes its own dimension in a "constraint space". The Jacobian matrix $\boldsymbol{J}$ projects accelerations from global forces into constraint space.

In general, the matrix on the left hand side of Eq. 4.8 is not square, making the problem under-constrained (or in some cases, potentially over-constrained). However, we can use d'Alembert's principle[65] to restrict the constraint forces to lie in the constraint space. Another method for arriving at the constraint equation is through the use of Lagrange multiplies. Consequently, the constraint-space forces are typically denoted with λ. The Jacobian transpose gives the relationship between a force applied in constraint space and force/torque applied in the full coordinate space: $\boldsymbol{f}_{ct} = \boldsymbol{J}_t^{\mathrm{T}} \boldsymbol{\lambda}_t$.

The vector, $\boldsymbol{\lambda}_t$, holds the generalized forces applied by each constraint on all the bodies involved in that constraint, whereas $\boldsymbol{f}_{ct}$ holds the sum of the constraint forces applied to each individual degree of freedom of each rigid body. The LHS of Eq. 4.8 can then be rewritten as $\boldsymbol{J}_t \boldsymbol{M}_t^{-1} \boldsymbol{J}_t^{\mathrm{T}} \boldsymbol{\lambda}_t$, where $\boldsymbol{J}_t \boldsymbol{M}_t^{-1} \boldsymbol{J}_t^{\mathrm{T}}$ is now a $n_c \times n_c$ positive semi-definite matrix.

It is informative to compare this equation,

$$\boldsymbol{J}_t \boldsymbol{M}_t^{-1} \boldsymbol{J}_t^{\mathrm{T}} \boldsymbol{\lambda}_t = -\frac{1}{h^2} \boldsymbol{\alpha} \boldsymbol{\phi}(\boldsymbol{x}_t) - \frac{1}{h} \boldsymbol{J}_t \dot{\boldsymbol{x}}_t - \boldsymbol{J}_t \boldsymbol{M}_t^{-1} \left(\boldsymbol{f}_{gt} + \boldsymbol{f}_{ut} \right) \tag{4.9}$$

with the equation used when satisfying a system using generalized coordinates (adapted from [51]):

$$\boldsymbol{\mathcal{J}}_t^{\mathrm{T}} \boldsymbol{M}_t \boldsymbol{\mathcal{J}}_t \ddot{\boldsymbol{\theta}}_t = -\boldsymbol{\mathcal{J}}_t^{\mathrm{T}} \boldsymbol{M}_t \dot{\boldsymbol{\mathcal{J}}}_t \dot{\boldsymbol{\theta}}_t + \boldsymbol{\mathcal{J}}_t \left(\boldsymbol{f}_{ut} + \boldsymbol{f}_{gt} \right) \tag{4.10}$$

The derivation for the generalized form is beyond the scope of this dissertation (see [36] or [51] for details). Equation 4.10 serves to show how the two methods are parallel and complementary. In this formulation, the Jacobian, $\mathcal{J}$, gives a linearization of how the bodies change when degrees of freedom change: $\mathcal{J}_{ij} = \frac{\partial x_i}{\partial \theta_j}$. In the generalized formulation, the constraining forces are implicit and the constraints cannot be violated by construction. Instead of computing gyroscopic forces, it is necessary to compute coriolis and centrifugal forces. The comparison between the methods is interesting and helps highlight some advantages presented by using maximal coordinates.

For our purposes, the biggest advantage of maximal coordinates is that the constrained degrees of freedom can be setup to be conflicting so there is no way to eliminate constraint error. Conflicting constraints might seem like a bad thing, but we make use of them in the next section to accomplish our goals. Because generalized coordinates require a function that converts from generalized space to Cartesian space, an equivalent approach to the one described below is not immediately obvious.

Returning to maximal coordinates, we will compress Eq. 4.9 down to

$$\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{w} \tag{4.11}$$

In general, the matrix $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^{\mathrm{T}}$ may be singular. It is very easy to end up with redundant or conflicting constraints. For example, a box resting on the ground may get a contact constraint at each corner. If each contact prevents interpenetration and sliding (i.e., applies friction) then the contacts constrain a total of 12 degrees of freedom on a single rigid body with only 6 degrees of freedom to be constrained.

Conflicting or redundant constraints can break the solver if not dealt with beforehand. The means for dealing with the conflict is clever. The physics engine softens the constraint, allowing it to "slip" proportional to the amount of force necessary to maintain it.

Because mass is always positive, the force, λ, applied to a particular constraint and the resulting constraint-space acceleration will have the same sign. Softening the constraint is therefore a matter of subtracting a scaled copy of λ from the desired acceleration (the right hand side): $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{w} - \boldsymbol{\beta}\boldsymbol{\lambda}$, where $\boldsymbol{\beta}$ is an $n_c \times n_c$ diagonal matrix of (typically small) non-negative values. This subtraction, of course, is equivalent to adding $\boldsymbol{\beta}$ to the LHS. Adding these small values to the diagonal of the effective inverse-mass-matrix makes the constraints seem lighter to the solver and moves the matrix away from singularity:

$$\left(\boldsymbol{J}_t\boldsymbol{M}_t^{-1}\boldsymbol{J}_t^{\mathrm{T}} + \boldsymbol{\beta}\right)\boldsymbol{\lambda}_t = -\frac{1}{h^2}\boldsymbol{\alpha}\boldsymbol{\phi}(\boldsymbol{x}_t) - \frac{1}{h}\boldsymbol{J}_t\dot{\boldsymbol{x}}_t + \boldsymbol{J}_t\boldsymbol{M}_t^{-1}\left(\boldsymbol{f}_{gt} + \boldsymbol{f}_{ut}\right) \tag{4.12}$$

The original programmers built soft constraints into the ODE simulation code. The variable, β, tunable for each constraint, is known in ODE as the *constraint force mixing* parameter (CFM). At first glance, the addition of these parameters may seem loose and unprincipled. However, correctly setting the parameters, α and β, changes a hard constraint into a simulated implicit spring with first order integration (see [34]). The significance of this innovation seems to be largely overlooked by the physics-based animation community, although Erin Catto recently gave a presentation pointing out some of the advantages of these stabilization parameters[17]. It is well-known that the formula for ideal damped spring force is identical to the formula for

PD control. However, connecting these two facts, namely that (1) ODE's constraints are mathematically equivalent to implicit damped springs and (2) damped springs are equivalent to PD controllers, has not been exploited, to our knowledge. This insight is key to the success of the methods presented here. One contribution of this work is to present a clear derivation showing that ODE's constraints are, in fact, stable PD controllers along with examples of how to take advantage of this fact. We proceed by discussing proportional-derivative control and the mass-spring-damper equation.

4.5 Implicit Simulated Springs

Proportional-derivative (PD) control is a common method used to compute forces that drive a system toward a target state. The PD control equation is the same as a mass-spring-damper system. There are two parameters, k_p and k_d, that determine what force should be applied to a degree of freedom at any point in time. The stiffness, also called proportional gain (k_p), specifies a force driving a degree of freedom toward its setpoint, $\bar{x}$ with strength proportional to the distance from the setpoint. The damping, also known as derivative gain (k_d), counteracts the current velocity, slowing the system down to avoid overshooting. When a system uses PD control to encourage a degree of freedom to move toward a target state, the control force f_{ut} at any instant in time is a function of the current position and velocity of the effective mass being controlled relative to its target:

$$f_{ut} = -k_p x_t - k_d \dot{x}_t \tag{4.13}$$

In a continuous time system, this controller is guaranteed to be stable as long

as k_d and k_p are non-negative. With zero damping ($k_d = 0$) the system oscillates in a sinusoidal wave pattern whose frequency is determined by the stiffness and mass and whose magnitude is determined by the initial conditions. With zero stiffness and positive damping, the velocity of the system decays exponentially with higher damping converging to zero more steeply. Discrete sampling of these forces, however, ruins the stability conditions. The potential for instability is apparent if we consider a mass m that only experiences damping forces. Using the semi-implicit Euler integrator, Eq. 4.2, we plug in the damping forces from Eq. 4.13 to get

$$\dot{x}_{t+h} = \dot{x}_t - \frac{hk_d}{m}\dot{x}_t = \left(1 - \frac{hk_d}{m}\right)\dot{x}_t \tag{4.14}$$

Time (t), mass (m), and damping (k_d) should all be non-negative values. It is clear, then, from this equation, that if $\frac{hk_d}{m} > 2$, the velocity will oscillate between positive and negative values and grow in magnitude. This oscillation rapidly causes the simulation to "explode" and is annoyingly common when using PD control. Overly stiff springs hit a similar limit with explicit discrete integration that causes them to gain energy and explode. Consequently, explicit PD control gains are tricky to tune. They must fall within certain limits that depend on the timestep and the effective mass experienced by the system.

The cause for this instability lies in the discrete integration which is similar to approximating the area under a curve as the sum of multiple rectangles computed forward from the present (Fig. 4.1). One solution is to solve for the forces *implicitly*. Implicit integration is similar to approximating the area under a curve with fixed-width rectangles that end rather than begin on the curve. Rather than overestimate,

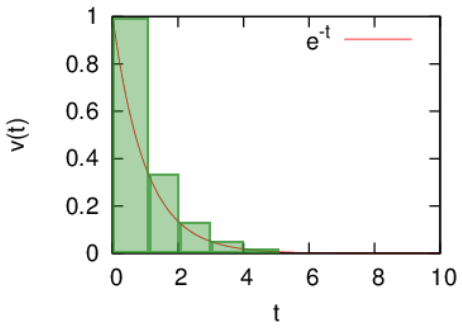


Figure 4.1: Explicit integration of damping forces is similar to the forward-method for approximating the area under a curve as a sum of rectangles. In this case it severely overestimates, leading to instability.

this method tends to underestimate the area under an exponential curve. The resulting system does not explode, although it tends to dissipate rather than conserve energy. The implicit form of the damped-spring-law depends on the integrator it is applied to. Being 'implicit', in this case, specifies that spring forces are computed from the *future* state of the system. Consequently, Eq. 4.13 becomes the following, (note the changed temporal indices):

$$f_{ut} = -k_p x_{t+h} - k_d \dot{x}_{t+h} \tag{4.15}$$

We do not know the future position or velocity, but using the integrator equations, Eqs. 4.3 and 4.2, we reframe Eq. 4.15 in terms of the current quantities and then solve for f_{ut} to get

$$f_{ut} = -\frac{k_d \dot{x}_t + k_p x_t + h k_p \dot{x}_t}{1 + m^{-1} h k_d + m^{-1} h^2 k_p} \tag{4.16}$$

If we analyze a pure damped system as before but using Eq. 4.16, we end up with

$$\dot{x}_{t+h} = \dot{x}_t - \frac{h k_d \dot{x}_t}{m + h k_d} = \frac{m}{m + h k_d} \dot{x}_t$$

With k_d now in the denominator, even an infinite damping gain is stable, corresponding to the damping force that completely eliminates the current velocity in a single timestep. This stability allows us to make PD controllers with extremely stiff gains.

Stability is a nice property for a controller or simulator to have. We now show that the α and β terms added to the constraint equation change them into implicit springs. To see the correspondence between Eq. 4.12 and Eq. 4.16, we consider a constraint that keeps a point mass at the origin along a single dimension: $\phi(x_t) = x_t$. The displacement function for this system has a trivial Jacobian: $J = 1$, meaning that $\lambda = f_c$. Assuming that external forces are zero, $f_g = f_u = 0$, Eq. 4.12 simplifies to

$$(m^{-1} + \beta) f_{ct} = -\frac{\alpha}{h^2} x_t - \frac{1}{h} \dot{x}_t \tag{4.17}$$

Assigning the α and β parameters[1] to be, $\alpha = \frac{hk_p}{hk_p + k_d}$ and $\beta = \frac{1}{h^2 k_p + hk_d}$, and isolating f_{ct}, Eq. 4.17 reduces to the implicit spring equation Eq: 4.16:

$$f_{ct} = -\frac{\left(\frac{hk_p}{hk_p + k_d}\right) x_t + h\dot{x}_t}{h^2 m^{-1} + h^2 \left(\frac{1}{h^2 k_p + hk_d}\right)} = -\frac{k_d \dot{x}_t + k_p x_t + hk_p \dot{x}_t}{1 + m^{-1} hk_d + m^{-1} h^2 k_p}$$

The consequence of this relationship is that every constraint in ODE can be thought of as an implicit spring. An important feature of this formulation is that the equations are solved simultaneously. To see the advantage of simultaneously computing spring forces, consider two springs both pulling on the same body in the same direction. If implicit spring forces are computed independently (i.e., using Eq. 4.16) and then

[1]These values are presented without derivation in the ODE user-manual: `http://ode-wiki.org/wiki/index.php?title=Manual:_All#How_To_Use_ERP_and_CFM`. Note that our formulation of β has an extra h in the denominator which is added automatically by ODE.

applied together, they pull too hard and become prone to overshoot. The system becomes unstable. When the implicit springs are solved simultaneously in the physics framework, the forces account for each other. Without this change the system would be very fragile. Softening the constraints to springs makes it so that we can solve a system that would otherwise be over constrained. We can add more constraints than there are degrees of freedom. It is not obvious how one might similarly soften constraints when using generalized coordinates, making it advantageous to use maximal coordinates for many tasks described in Chapters 5 and 6.

4.6 Solving with Complementarity Conditions

For simplicity, we compress Eq. 4.12 down to $\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{w}$. When $\boldsymbol{A}$ is non-singular, we can solve for $\boldsymbol{\lambda}$ by inverting, or preferentially, using a fast, numerically-stable solver such as a Cholesky decomposition. Some constraints, however, come with additional conditions that need to be solved with extra machinery. In simulation literature, these are known as inequality constraints. For example, a contact constraint keeps two bodies from moving towards each other by defining an error function that is the separation of the contacting surfaces in the direction of one of the surface normals. If the surfaces are overlapping, then the error function has a negative value and a positive constraint force will accelerate the surfaces apart. This acceleration is as it should be. However, the linear system also applies forces to correct positive error; so the same constraint would also prevent the surfaces from separating.

The solution to this problem is to limit the amount of force available for

satisfying the constraint. A contact constraint, in particular, limits the force to be non-negative. Contact friction constraints are limited on both sides to be proportional to the contact normal force. This limitation places upper and lower bounds on the constraint force variable: $\boldsymbol{\lambda}_{\text{lo}} \leq \boldsymbol{\lambda} \leq \boldsymbol{\lambda}_{\text{hi}}$, allowing constrained bodies to accelerate without bounds if the force necessary to hit the acceleration target falls outside of the limits. In ODE, the result is three possible conditions to satisfy a constraint:

1. $\boldsymbol{a}_i\boldsymbol{\lambda} = w_i$ with $\lambda_i \in [\lambda_{i\text{lo}}, \lambda_{i\text{hi}}]$,
2. $\boldsymbol{a}_i\boldsymbol{\lambda} > w_i$ with $\lambda_i = \lambda_{i\text{lo}}$, or
3. $\boldsymbol{a}_i\boldsymbol{\lambda} < w_i$ with $\lambda_i = \lambda_{i\text{hi}}$

where $-\infty \leq \lambda_{i\text{lo}} \leq 0 \leq \lambda_{i\text{hi}} \leq \infty$.

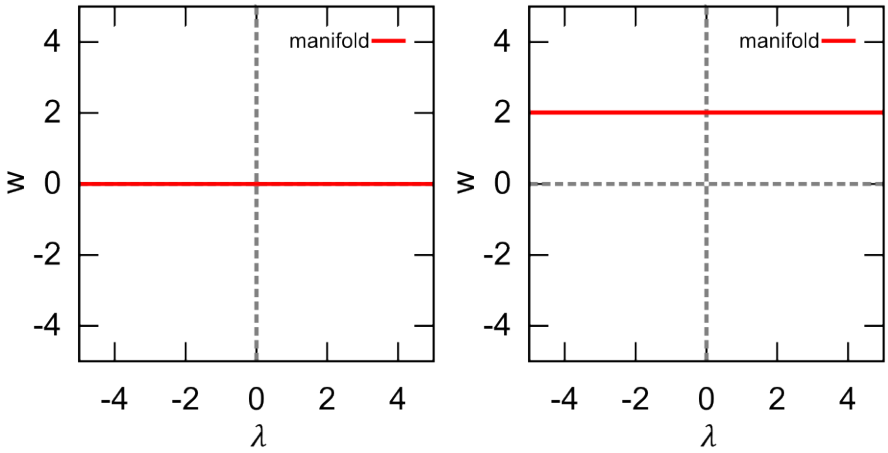
A linear solver cannot handle these extra conditions on the constraint forces. To solve this type of system, physics engines employ a mixed Linear Complementarity Problem (mLCP) solver. ODE offers two different solving methods for satisfying constraints under limited-force conditions. One method, known as Projected-Gauss-Seidel, solves constraints iteratively and accumulates the effects[16]. Iterative methods tend to be faster, but also tend to be inaccurate when the system is near-singular or ill-conditioned. Simulated humanoid systems, particularly with two feet on the ground, tend to behave badly with this faster solver. The slower, pivot-based method, follows the algorithm presented by Baraff[8]. Baraff's method is still easily fast enough for our purposes.

Each row in matrix $\boldsymbol{A}$ represents a constraint. The corresponding values of $\boldsymbol{w}$ and $\boldsymbol{\lambda}$ represent a "target" acceleration along the degree of freedom constrained by that row and the generalized force used to achieve it. For the i^{th} row of $\boldsymbol{A}$, the diagonal element, a_{ii}, behaves like the inverse mass of the constraint. A force, λ_i, imposes an acceleration of $a_{ii}\lambda_i = w_i$ within the constraint error-space. The rest of the elements in a row of $\boldsymbol{A}$ encode the force's effects on other constraint dimensions. A change in the i^{th} constraint force λ_i affects the j^{th} constraint space by accelerating it according to $\delta w_j = a_{ij}\delta\lambda_i$. The order of the constraints is arbitrary and they can be rearranged as long as every row-swap is accompanied by the corresponding column-swap that maintains the proper symmetry.

Baraff's solving algorithm (based on Dantzig's simplex method) takes advantage of this arbitrary ordering by dividing constraints into different sets: a satisfied set S, a limited set N, and an unaddressed set U. All constraints fit into one of these categories. The first step in finding a solution is to reorder and satisfy all the unlimited constraints, without considering the rest, using a basic linear solver. The resulting system looks like

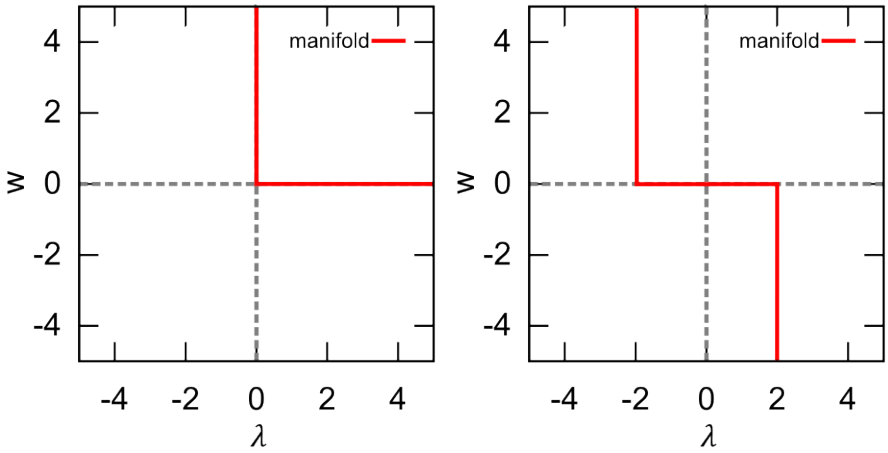
$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{12}^{\mathrm{T}} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_1 \\ \boldsymbol{A}_{12}^{\mathrm{T}}\boldsymbol{\lambda}_1 \end{bmatrix} \tag{4.18}$$

Set S holds the rows of $\boldsymbol{A}_{1i}$. Set U holds the rest. At this point it helps to look at some figures to see what is going on. Each constraint's target conditions can be represented as a piecewise line through force-acceleration space (Fig 4.2). We will call this multi-segmented line the target manifold for each constraint. Viewing constraints this way is another contribution of this work. The diagonal element of $\boldsymbol{A}$ associated



(a) Holonomic: a rigid constraint demands the relative acceleration between two bodies be zero.

(b) Motor: an unlimited motor constrains the relative acceleration to take on some value.



(c) Contact: a contact constraint prevents bodies from accelerating toward each other, but applies no force to prevent them from separating.

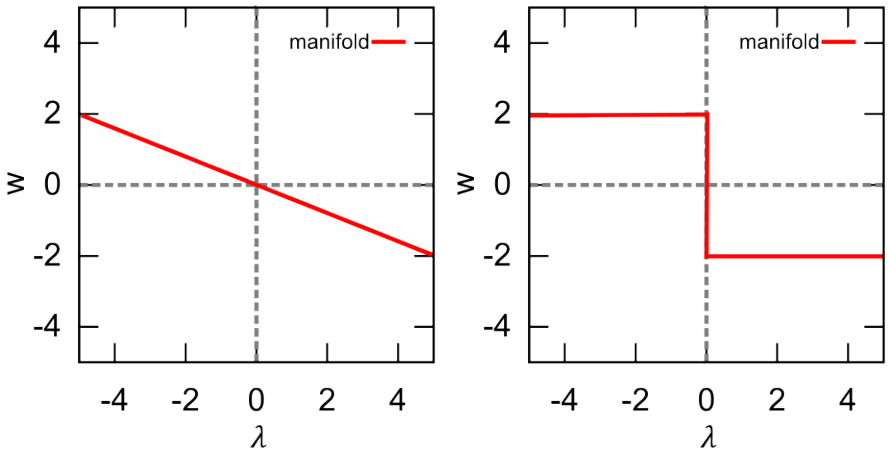
(d) Friction: a friction constraint prevents relative acceleration until a force limit is reached.

Figure 4.2: Each constraint on a single degree of freedom can be thought of as a monotonically decreasing, piece-wise linear target manifold through acceleration-force space.

with the constraint gives the slope of a line through the origin that represents the relationship between force (λ) and actual acceleration (A_{ii} is the effective inverse-mass of the i^{th} constraint). The solver seeks to find a joint solution so that, for all rows of $\boldsymbol{A}$, the pairs of (λ_i, w_i) fall on the acceptable manifold. Forces from other constraints move the entire manifold up or down relative to the origin.

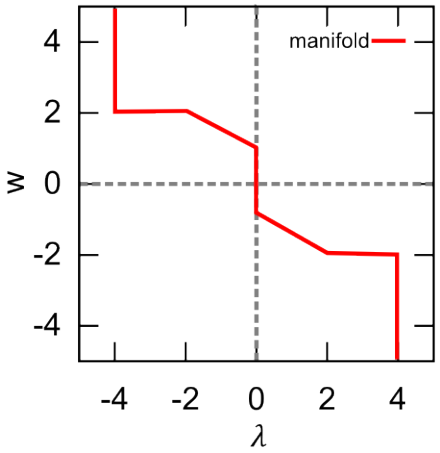
The β parameter takes the horizontal portion of the target manifold and tilts it so that when bigger forces are used, there is a lower target acceleration. Hence the constraint is spring-like. The vertical portions of the constraint represent places where the constraint has hit its force limits. No additional force can be applied by that constraint; so the acceleration must be allowed to increase freely. Otherwise, the constraint would be "obligated" to apply more force to try to get closer to its target acceleration.

Constraints are addressed one-at-a-time. When dealing with ground contact force without softened constraints, once the solver found a sufficient force to keep a body from penetrating the ground, any remaining ground contact constraints would have nothing to do, resulting in inappropriate distribution of ground forces. With spring-like constraints, if one contact constraint supporting a body reaches its target force/acceleration, a second, redundant contact constraint will see whatever distance remains between the current acceleration and the target. Forces applied by the second constraint attempting to reach its target push the target manifold of the first constraint toward the origin. The force required to achieve the first constraint's target decreases until the forces balance appropriately. The balancing forces make it possible to more accurately compute inverse dynamics forces.



(a) Elastic: when β is non-zero for a constraint row, the target acceleration decreases proportional to the force necessary to achieve it.

(b) Deadzone: using the same mechanism, we can introduce constraints that keep the relative acceleration of two bodies within a *range*.



(c) Complex: we can combine multiple linear pieces to create novel, complex constraints.

Figure 4.3: Adding a small value to the diagonal elements of the projected inverse mass matrix turns the constraint into a spring. Viewing constraints as piecewise linear targets provides insights into how to make more complicated constraints consisting of additional piecewise segments.

The algorithm for solving the mLCP progresses through each unaddressed constraint, one at a time, and finds the change in forces that will satisfy the new constraint without moving any of the current constraints off their piecewise target. Each iteration of the algorithm draws a new constraint from the unaddressed set U and addresses the change in force, λ, that will satisfy the new row without pushing any previously addressed rows off their manifold, until the new row can be added to S or N. In the process other rows may change between sets S and N, but each row remains on its target manifold in acceleration/force space.

Consider this partitioned matrix:

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{a}_{13} \\ \boldsymbol{A}_{12}^{\mathrm{T}} & \boldsymbol{A}_{22} & \boldsymbol{a}_{23} \\ \boldsymbol{a}_{13}^{\mathrm{T}} & \boldsymbol{a}_{23} & a_{33} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_1 \\ \boldsymbol{v}_2 \\ v_3 \end{bmatrix} \tag{4.19}$$

Adding a new force, λ_3, will change the accelerations of the other constraints. Accelerations of constraints at their limit are allowed to change, but those in set S must remain at their target. So we find the $\delta\lambda_3$ that moves v_3 toward w_3 and find the simultaneous $\delta\boldsymbol{\lambda}_1$ that keeps constraints in S satisfied. The constraint force takes the largest step that will not push any row out of its set. This step will either satisfy the constraint or move another constraint to an intersection point on its manifold. We then pivot the sets around and continue until all of our rows are in S or N. For additional detail, see [8].

Recognizing that the solver deals with each constraint target as a piecewise linear manifold provides useful insight into how the simulation mechanism can be improved. One obvious extension is to increase the number of linear segments in the target manifold beyond three (Fig. 4.3). This innovation becomes obvious when

constraints are considered as target manifolds rather than Lagrange multipliers. With a multi-segment target manifold, it is possible to create a spring-like constraint that is loose near its setpoint, but then becomes stiffer.

We can make spring constraints that get more or less stiff as additional force is required. We can also introduce constraints with "deadzones" in their PD control (Fig. 4.3). This type of constraint is particularly interesting because it allows us to introduce controllers that only come into play when a dimension of interest drifts out of an acceptable range. This type of controller takes inspiration from the idea of "uncontrolled manifolds" in human motor control theory[91]. With this constraint acting as a controller, if a perturbation will not hurt performance, the controller does nothing.

From deadzone controllers, we can introduce novel constraints with secondary targets. A constraint whose forces and accelerations fall within acceptable tolerances has flexibility to "help" another constraint that has reached its limit. For example, we can specify a target range for the knee, hip, and ankle joints of a simulated character. When these leg joints fall within their stated ranges, they can be allowed to pursue a secondary goal such as keeping the torso upright or at a given height. This type of constraint can serve as a method for reducing the need for unrealistic residual forces. Removing residual forces implies deviating from original kinematic data. Constraints with secondary targets make it intuitive and clear how this deviation will occur can be extremely beneficial when using the simulation engine for analyzing or synthesizing movement data. We have created and submitted code for allowing controller constraints with a deadzone in acceleration space. Full implementation

of secondary targets for constraints is still in progress. It promises to be useful for creating intelligent constraint-based controllers.

4.7 Summary

Significant programming effort has made ODE a fast, robust system for simulating constrained, physical systems. We have presented details on how it works and how it differs from other approaches using generalized coordinates and shown a useful derivation for the equations of motion. Treating every constraint as a spring helps stabilize complex systems such as simulated humanoids. Understanding the ODE solver and the mLCP system it works on allows us to build novel constraints with useful properties for controlling humanoid systems. In the course of this research, we have implemented and contributed several new types of joint constraints in the ODE code base. These can be used to simulate systems that were previously unavailable. We have also identified and fixed certain bugs in the ODE codebase and made other contributions designed to improve the stability of the solver in various conditions that arise when simulating humanoid characters.

Along the way, we have seen additional potential contributions to ODE that deserve attention in the future. Partial implementation of constraints with additional piecewise linear segments has been submitted to the ODE codebase. Full implementation of secondary-target constraints remains an outstanding project that can allow underconstrained joints to "help" overconstrained joints achieve target accelerations. These new joints can potentially make it possible to make an underconstrained humanoid model remain upright and stable without using non-realistic forces. Other

future contributions to the ODE codebase may include methods for handling disparate mass ratios between constrained bodies. In current simulation software, rigid bodies, held together by constraints, must have relatively similar masses or else the constraint matrix becomes ill-conditioned and unstable. The interpretation of the equations of motion presented here suggests that it might be possible to condition the constraint matrix by using different "scaled units" to represent the positions and velocities of bodies of different mass.

The field of numerical simulation has been fascinating to study, and we intend to continue working to develop this tool. Forward dynamics simulation software already can perform many impressive feats. The insights derived from our analysis of ODE and minor improvements made to the code allow us to accomplish several important tasks presented in the next two chapters. Principally, combining the known fact that ODE constraints are equivalent to implict springs with the fact that damped springs are equivalent to PD controllers allows us to deliberately use ODE's constraint solver as a controller instead of just for forward dynamics. Because the springs are implicit and all the constraints are solved simultaneously, the solver makes a very stable controller, even with stiff gains.

Chapter 5

Simulation-based Movement Analysis

Understanding the mathematics underlying physical simulation allows us to take advantage of its strengths. In particular, recognizing that constraints behave like implicit springs is extremely useful. The parameters that soften constraints into springs stabilize simulation, pushing a constrained system away from singularity and reducing constraint error. This chapter shows how the same mechanism can be useful for analyzing movement data. Using the constraint solver as a controller makes it extremely easy to integrate inverse kinematics and inverse dynamics analysis into real-time applications such as games or behavioral experiments that already need physics computations for forward simulation. The solution is robust, fast, free, and allows you to work with a single character model rather than needing to use a specific model provided by a commercial package.

Movement data, as captured during an experiment in virtual reality (Chapter 3) or a standard motion capture session, requires further analysis transforming it from trajectories of points through space to descriptions of character kinematics and dynamics to be useful. The initial output of marker-based motion capture is the position of the markers over time. These points are the locations of optical markers attached to clothing or skin. We can figure out the human's pose, the relative position

and orientation of his or her body parts in space, by fitting a model to the marker data.

We begin by combining the physics engine with a gradient-based learning approach to find where captured markers sit on the different body parts of a character as well as where body parts attach to each other. With this information we can find the model pose (or joint angles) over time by constraining a physically simulated model to satisfy marker constraints, joint constraints, and other relevant constraints such as non-penetration with the ground. Provided with a sequence of poses, it is straightforward to then constrain the model to transition through the pose sequence using internal joint torques. This process is known as "inverse dynamics".

Scientific literature covers these cases separately with different algorithms to handle each. In our approach, all of these tasks are accomplished using the simulation engine. Our method has several advantages. First it can be easily implemented in a single robust framework of the physics engine. Using the physics engine for multiple tasks allows a single character model to be used from start to finish, rather than being forced to use the model built in to a commercial package. Second, the method is fast. The simulation engine is designed for performance, making it possible to analyze movement in real-time and create interactive experiments with stimuli dependent on the feedback results. Third, the software is free. Freely accessible code, such as ODE, is useful because it facilitates comparison and collaboration in research. Fourth, the method easily deals with multiple ground contacts and noisy data that can be challenging to related approaches. Kinematic loops do not require any special treatment. The method is robust even to large perturbations making data

dynamically inconsistent. Finally, the tunable parameters, couched in the physics framework, are intuitive. It is more straightforward to specify the importance of a constraint in terms of force and mass rather than arbitrary gains and weightings. We illustrate these advantages by using ODE analyze and reproduce movement recorded from optical motion capture.

We have produced an example implementation using the physics engine as a multi-purpose tool for analyzing movement data captured through interaction with the virtual environment (Figure 5.1). Using this tool, it is possible to interactively fit a model to marker data, scrub through data, dynamically adjust parameters to test different effects, and visualize the results of kinematic and dynamic analysis.

5.1 Model Construction

Our techniques use a simulated model of the human whose movement is analyzed. The first order of business is to build a physical model capable of representing human movements. The accuracy of the model influences the outcome of the analysis. The humanoid model is a collection of rigid bodies connected by joint constraints. We present here a method for using marker data to help determine the dimensions of the model segments and where markers attach to the model.

The technique for fitting a model to data begins with a character model that serves as a template, Fig. 5.2, providing the number of body segments and topology of the model. We further require that labeled markers be assigned to specific model segments. It may be straightforward to derive these using a technique such as in [26, 58]. However, it is also not difficult to do by hand. It would become tedious if

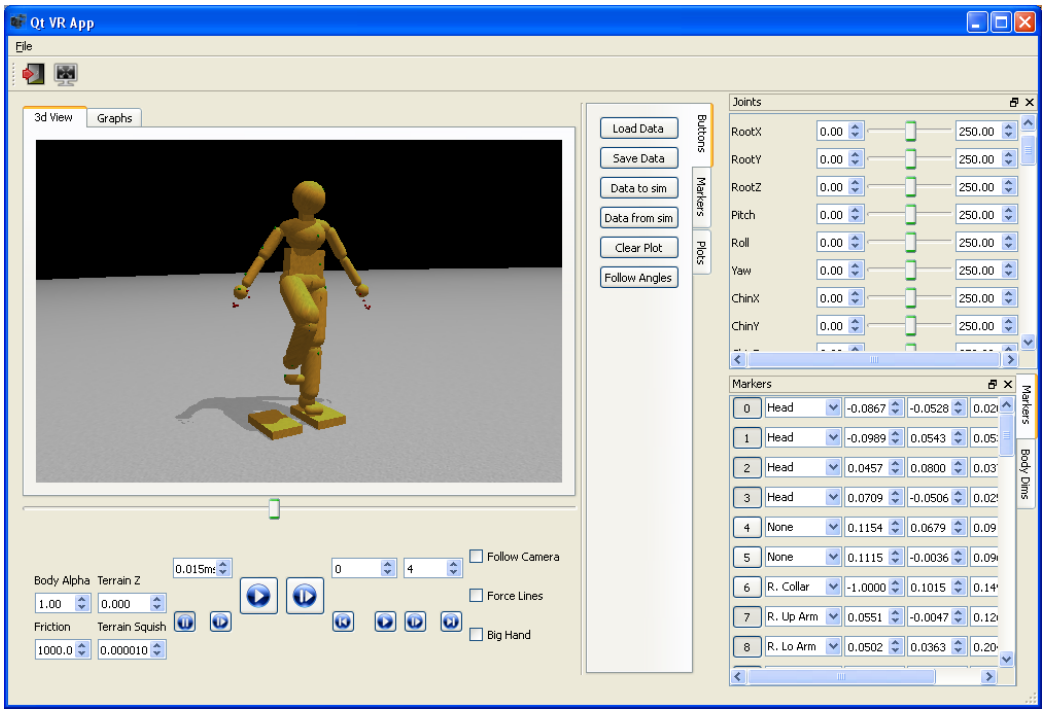


Figure 5.1: Our analysis tools use the physics engine to compute inverse kinematics and inverse dynamics. They also support various visualizations of relevant data and control for analyzing and producing physically-based movements.

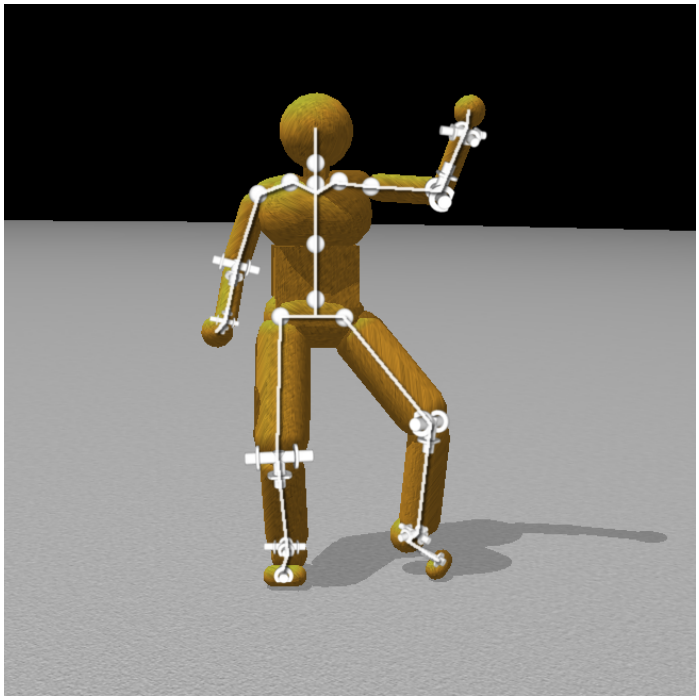


Figure 5.2: Starting template for fitting a model to marker data. This template describes which joint segments attach to others and also specifies some limits on angular degrees of freedom.

one had to go through the process for many different models. Fortunately, the motion capture suit typically puts the markers on the same body segments, even if they are in slightly different places and the body segments have different dimensions.

The model consists of n_b rigid bodies connected by n_j joints. In this case, each joint consists of three to five constraints. Each joint connects two rigid bodies with anchor points (center of rotation) defined in the reference frame of both bodies. The joint constraints keep the anchor points relative to the two bodies together in the global frame. If bodies b_j and b_k are connected, a joint constrains them together at a common point. The joint anchor relative to body b_j is $\tilde{\boldsymbol{c}}_{jk}$. The anchor for body b_k is $\tilde{\boldsymbol{c}}_{kj}$. The joint constraint drives these points together in the global frame, creating three constraint rows:

$$\boldsymbol{\phi}_{jk} = \boldsymbol{R}_j \tilde{\boldsymbol{c}}_{jk} + \boldsymbol{x}_j - \boldsymbol{R}_k \tilde{\boldsymbol{c}}_{kj} + \boldsymbol{x}_k$$

The locations of these anchor points determine the segment dimensions (bone lengths) of the character model.

Markers, each assigned to a specific rigid segment, represent a point on the human's body. We seek anchor points that allow markers to remain approximately stationary relative to their assigned body segment. It is generally impossible to find such a configuration exactly (without creating an unreasonable number of body segments) because of soft-tissue artifacts (STAs). Skin and joints are not actually rigid. They stretch and give as muscles pull the bones. Modeling the body in maximal coordinates provides a way to model STAs explicitly.

Given a pre-defined model topology and markers assigned to specific model

segments, we seek to find the joint anchor points between segments and the marker attachment points relative to the model segments. If the i^{th} marker is assigned to the j^{th} rigid body ($\boldsymbol{p}_i \rightarrow b_j$) at relative point $\tilde{\boldsymbol{s}}_{ij}$, we model the marker's attachment as a three dof constraint:

$$\boldsymbol{\phi}_{ij} = \boldsymbol{p}_i - \boldsymbol{R}_j \tilde{\boldsymbol{s}}_{ij} - \boldsymbol{x}_j$$

The process models markers from an arbitrary point in time as infinite point masses. As bodies of infinite mass, constraint forces do not affect the markers' trajectories but only the bodies they are anchored to. Initially, markers are anchored at $\tilde{\boldsymbol{s}}_{ij} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. This mapping attaches the marker to body b_j's center of mass.

Clearly, this mapping is a very rough estimate of the marker attachment points on the model segments, but it is sufficient because of the flexible nature of constraints in the simulation software. Setting the *CFM* parameter of the marker constraints to $\beta = 10^{-3}$ and setting the model joint constraint *CFM* to $\beta = 10^{-5}$ makes the body segments hold together tightly, while still allowing the markers to pull the body into shape. Several timesteps of simulation allow the model to relax to a fixed pose. We then take the markers in their current configuration and reattach them to their respective segments. Relaxing the marker attachments this way greatly improves the fit for this particular frame of marker data. Iteratively repeating this process with multiple frames of marker data, we therafter update the marker attachment points by some learning rate, η_m: $\tilde{\boldsymbol{s}}'_{ij} = (1 - \eta_m)\tilde{\boldsymbol{s}}_{ij} + \eta_m \boldsymbol{R}_j^{\mathrm{T}} (\boldsymbol{p}_i - \boldsymbol{x}_j)$. Gradually updating attachment points, using different frames of data, effectively descends the

error gradient of the marker positions relative to the body:

$$\min_{\tilde{\boldsymbol{s}}} \sum_{t=1}^{T} \sum_{i=1}^{n_m} \|\boldsymbol{p}_i - \boldsymbol{R}_j \tilde{\boldsymbol{s}}_{ij} - \boldsymbol{x}_j\|$$

The decrease in marker error affected by model dimension error. Conveniently, joint anchor constraints behave the same as the marker attachment constraints. With an arbitrary frame of marker data and using a marker *CFM* of $\beta = 10^{-4}$, if the markers constraints cannot be satisfied, they will pull the joint anchors apart slightly. For each joint we find a new common anchor point in the global frame by taking the average between the two unsatisfied anchor points that the joint constraint is trying to pull together. We then move the anchor points toward that point according to learning rate η_l:

$$\tilde{\boldsymbol{c}}'_{jk} = (1 - \eta_l)\tilde{\boldsymbol{c}}_{jk} + \eta_l \boldsymbol{R}_j^{\mathrm{T}} \left(\boldsymbol{R}_k \tilde{c}_{kj} + \boldsymbol{x}_k - \boldsymbol{x}_j\right)$$

For any one frame, errors will cause the markers to stretch from their attachment points and joint anchor points to stretch apart from each other. Both the marker attachment points and the joint anchors can be updated simultaneously to decrease the error for that frame. However, the local solution that perfectly satisfies one frame, may make another frame worse. This presents an obvious gradient descent approach to finding the joint anchors and marker attachments: using several frames, compute an average adjustment to the marker attachments and joint anchors that reduces the error. Make the adjustment to both anchors and attachments and then iterate. It may be advisable to employ the standard machine learning practice of a validation set to ensure that the error continues decrease and avoid over fitting. This

technique relies on spring-like constraints made possible in maximal coordinates as provided by the derivation in Chapter 4.

Although this method could easily be automated, that is left for future work. In practice, the research did not rely on very many different models and so we added a mechanism for relaxing the marker attachment points and joint anchors with the click of a button in the graphical user interface (Fig. 5.1). With a new data set, a handful of iterations proved sufficient to produce a reasonable model with marker attachments that fit the data well-enough to be used for further analysis. This algorithm does not address joint limits on range of motion. These can also be learned[109], but in our case, the range of motion for each joint is set *a priori*. After determining segment lengths, we set other segment dimensions as appropriate to fit against the markers. Mass properties for each segment assume uniform density by volume.

For the data presented here, the model is fit to the subject's dimensions and joint-range-of-motion is constrained to approximate the subject's flexibility. Although the task of finding body pose is normally considered a purely kinematic pursuit, the model segments have mass properties that influence the outcome in our approach. We assign mass to each segment by assuming a uniform density of water ($1000\frac{kg}{m^3}$) for the volume associated with each rigid body. In our data, this constant density assumption produces a model with a total mass of approximately $80kg$, roughly matching that of the subject. For increased fidelity, as required for clinical biomechanics research, we would employ more sophisticated techniques for a better approximation of mass distribution in the model. Interestingly, however, even this low fidelity model is sufficient to produce high-quality data that compares favorably with data gathered

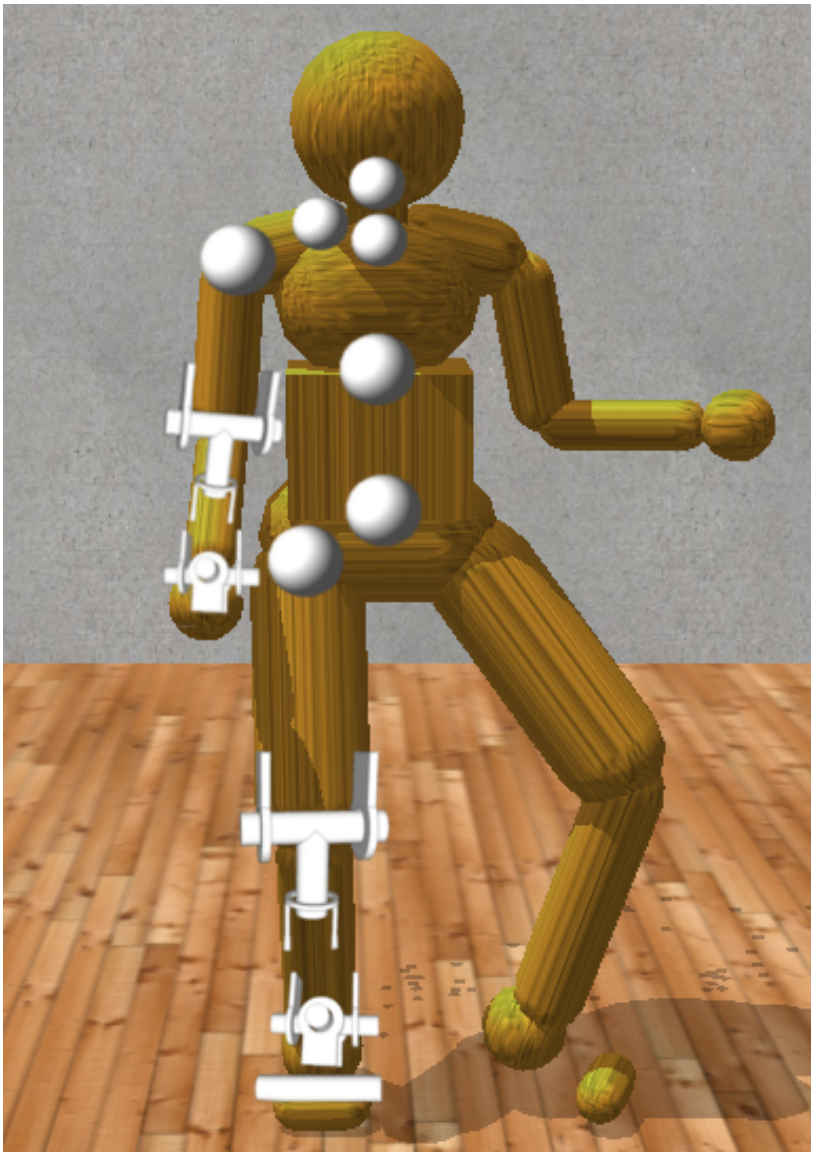


Figure 5.3: Our humanoid model has 48 internal degrees of freedom. All joints are constrained to stay within semi-plausible joint limits and use limited torque. We adjust limb dimensions and articulation points to fit marker data.

from other sensors.

Our model consists of 21 separate rigid bodies connected by 20 joints (Fig. 5.3). The unconstrained bodies have $6 \times 21 = 126$ total degrees of freedom. The joint anchors, constraining pairs of bodies to share a common center of rotation, remove $3 \times 20 = 60$ degrees of freedom. It is important to remember, however, that we do not treat joint connections as holonomic (perfectly rigid) constraints, but rather as very stiff springs that hold body parts together like tendons and muscles. We further constrain the relative orientation of some bodies by using universal joints for the elbows, wrists, knees, and ankles and hinge joints to connect the toes to the heels. Universal joints restrict one angular degree of freedom; e.g., when the arm is bent at the elbow, the forearm cannot rotate around the principal axis of the upper arm unless the upper arm itself rotates. However, the forearm can rotate at the elbow around its own principal axis (modeling the twisting movement of the radius and ulnar bones). These constraints on relative orientation remove an additional 12 degrees of freedom. All other joints are left as ball-and-socket joints with three angular degrees of freedom: hips, shoulders, collar-bones, upper-neck, lower-neck, upper spine, and lower spine. This arrangement of joints leaves a total of 48 unconstrained internal degrees of freedom and 6 external degrees of freedom. We add stiffness constraints to all of these so that the body is not floppy.

5.2 Pose Fitting

Although various commercial packages provide different methods for converting marker trajectories into sequences of character poses, they come with various

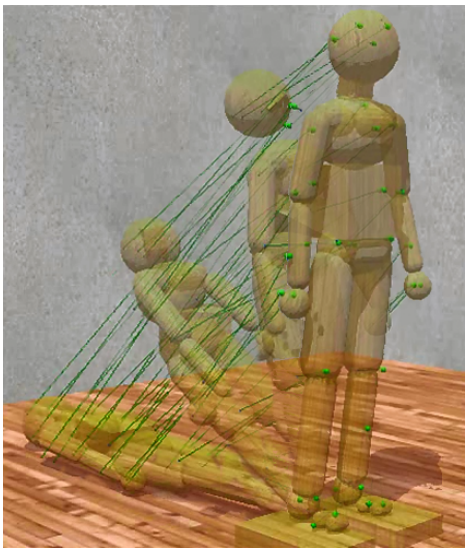


Figure 5.4: Markers pull a simulated character model into place.

inconveniences. They can be time-consuming, expensive, or confusing to use, may require a specific character model, or produce unintuitive results. We present an approach related to [30] and [125] that is free, fast, uses intuitive parameters, and allows the user to fit markers to whatever model they wish.

Our method uses the physics engine to constrain a character model to fit marker data and other constraints. As above, we model markers as infinitely massed points attached to the character model. We also program each joint to pursue a "target" orientation by creating a joint constraint with weak, limited torque. Balancing internal joint targets and external marker data can be seen as a type of "Bayesian" approach where markers serve as evidence for what the current pose should be while internal joint stiffnesses bias the model toward a "prior" pose. The purpose is to find the position and orientation of all body segments, given a frame of marker data.

From the global position and orientation of the different body segments, it becomes a simple matter to compute relative orientations (joint angles).

The internal degrees of freedom are limited by range of motion constraints. The elbows and knees cannot bend backwards. All other joints have similar range-of-motion limitations based on the subject's flexibility. Furthermore, we assume that each joint has a "target state", a preferred relative orientation between its connected bodies. These preferences can be thought of as "muscle stiffnesses" and are modeled as weak constraints with limited force. Joint limits and stiffnesses serve as a prior over possible poses so that in the absence of any marker data at all, the model still takes on a pose. Consequently, every internal degree of freedom is constrained to some degree. The joints in the model shown in Fig. 5.2 result in 120 constraints on the full system.

These constraints hold the model together and give it a default pose. Marker data pull the model from the default pose into a new pose (Fig. 5.4). Each marker is associated with a body segment and has an associated attachment point. For a given frame of marker data, we model each marker as an infinite mass at point $\boldsymbol{p}_i$. We then connect the attachment point, $\tilde{\boldsymbol{s}}_{ij}$ of each body marker to its associated body segment with a ball-and-socket joint constraint. A total of 36 markers, which do not contribute any degrees of freedom because of their infinite mass, attach to the character model, adding an additional $3 \times 36 = 108$ constraint dimensions.

Finally, collisions between the ground and the feet also influence the model pose. Each foot can form up to three contact points with the ground. Inequality constraints at these points prevent penetration with the ground. When both feet

are firmly on the ground, all markers are actively pulling the body into a pose, all joints are holding the body together, and joint limits and stiffnesses are biasing the relative orientation of the bodies, there are 234 active constraints in our model, even though there are only enough rigid bodies for 126 degrees of freedom. Conventional wisdom states that having twice as many constraints as degrees of freedom is a bad idea. However, our experiments show that the overconstrained system is easily handled in real-time with 60 fps data. The majority of the system is sparse and solved very quickly with a Cholesky decomposition, leaving only a few iterations and pivots necessary to solve the inequality portions of the system (see Chapter 4). The key is that the system is designed to be fast and robust.

This approach is simple intuitive: attach markers to the model with springs and then drag the body along. Compared to solutions minimizing squared marker error this solution can be more robust. Squared error makes noisy markers are a big problem. A blip in the motion capture that causes a marker location to jump results in a big squared error, kinking the skeleton. A spring, on the other hand increases its pull linearly with distance. All of the other springs pulling on the body, along with the body's own inertia, tend to neutralize a single noisy point. Other pose estimation approaches require an abundance of markers to fully specify the position and orientation of each body segment. The ability to bias the skeleton solution with a stiffness prior allows far fewer markers. A disadvantage of this approach is a slight decrease in overall accuracy caused by the springs lagging behind the markers. This problem can be reduced by allowing multiple iterations for convergence. However, our goal was real-time processing sufficient for use in virtual reality experiments and

other interactive applications. A single pass provided sufficiently accurate analysis in our tests to be used for interactively animating an avatar or computing pose to satisfy end-effector constraints.

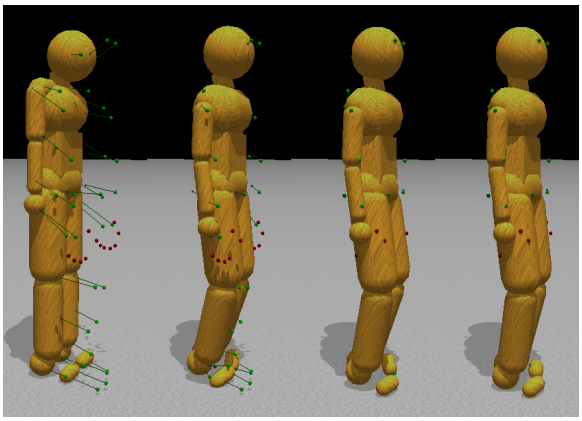
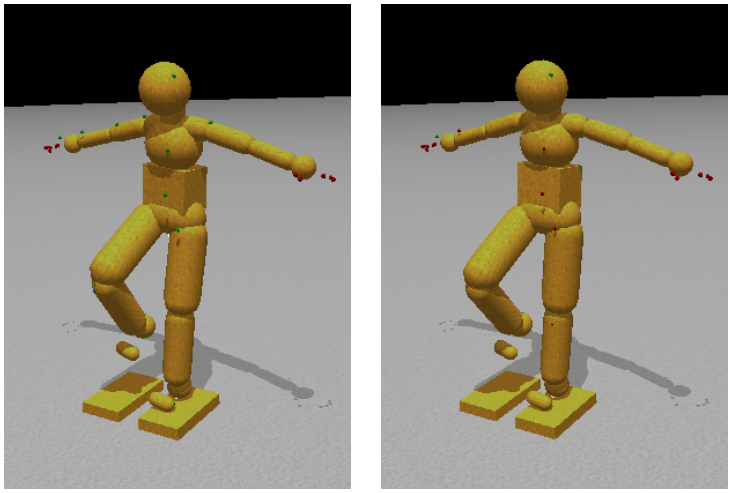


Figure 5.5: Simulated markers attach to the humanoid model through ball-and-socket joints and pull the body parts into place, subject to model joint constraints.

When the markers move, the constraints drag the character along with them (Figure 5.5). The CFM term allows a constraint to slip proportional to the amount of force that would be required to maintain the constraint. For the regular internal body joints and contact constraints, we use a CFM value of 1×10^{-5} while for the constraints between markers and body parts we use 1×10^{-4}. Both of these values represent very stiff springs although they are different by an order of magnitude. This stiffness stabilizes the simulation by allowing the markers to stretch slightly from their mapped locations in the event that the marker constraints are not compatible with the character model.

In tests, we found that for many movements, with suitable internal stiffness, it is only necessary to control location of the head, hands, and feet. Figure 5.6 shows



(a) Body configuration using all marker constraints. Note the similarity to the sparsely constrained pose.

(b) Body configuration using constraints on only the head, hands, and feet.

Figure 5.6: In many cases, the pose found using a full set of marker constraints is quite close to that found by a sparse set of constraints. These two images show almost no differences between using a full or a sparse set of marker constraints.

a qualitative comparison between a pose found using the whole marker set and one found using only head, hands, and feet. We set all joints to attempt to use a small torque to move toward a default pose (arms and legs straight down). Joint limits on the knees and elbows and general joint stiffness naturally bias the physics engine to find a pose that is very close to the fully constrained pose. Body inertia and joint stiffness naturally clean up minor noise and occlusions in the captured marker data. We can use the resulting joint angles to then solve the inverse dynamics.

5.3 Inverse Dynamics

It can be useful to know how much effort is required to accomplish a particular movement. Robot control may rely on knowing torques to apply at each joint. In clinical or experimental settings, a measure of effort exerted during a particular action can be useful for medical reasons. Given a kinematic sequence of body poses, the physics engine once again accomplishes our desired purposes with minimal effort. The engine must compute constraint forces as a matter of course. We simply need to frame the constraints correctly to compute the desired measurements.

This process is straightforward. Given the current joint angle and the desired joint angle for the next frame, we constrain the relative angular velocity of the body parts as necessary to achieve the target orientation on the next frame. Contact constraints are again necessary to prevent ground surface penetration. The ODE physics library handles the constraints by solving Eq. 4.12. The torques and forces, $\boldsymbol{\lambda}$, used to satisfy each constraint are solved in the process and made available by the software library.

When computing inverse dynamics, we initialize the character model an starting dynamic state. The initial state can be found from two consecutive frames of kinematic pose data. We set the model pose using the second frame of data, and set the linear and angular velocity of each body by taking the finite difference between the two frames (and dividing by the timestep). Computing velocity through finite differences is appropriate for a physics engine using first-order semi-implicit Euler integration.

We find the torques between the second and third frames of pose data by once again using the finite difference between poses to compute angular velocities that will move the model from the second to third pose. Differentiating again, this time between the current and future velocity gives a target acceleration that becomes a constraint on the model. The primary difference between this step and the previously discussed method for finding pose from marker data is that there are no marker constraints dragging the body into place and the internal muscle stiffness drives the model toward a target pose on each frame instead of toward a 'default' pose. Because there are fewer constraints in play, we can use stiffer muscles ($\beta = 10^{-8}$) but limit the absolute forces the muscles can apply. Force limits prevent muscle forces from being unreasonably large.

Again, in this case, we can use the relative spring stiffnesses to express our confidence in our measurements. We use very stiff springs ($\beta = 10^{-10}$) to keep the model segments together. We use looser constraints to keep the feet from penetrating the ground ($\beta = 10^{-5}$) and to constrain the model to adopt the appropriate pose ($\beta = 10^{-8}$).

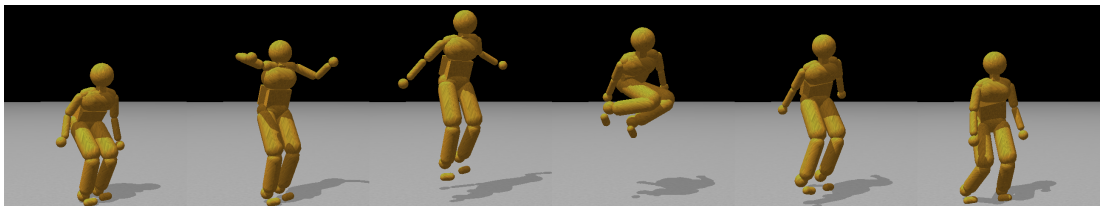


Figure 5.7: A jump sequence reproduced with physics-engine-based inverse dynamics. The jump height is achieved completely from ground forces, with small residual torques ($\leq 100Nm$) keeping the model from tipping over.

Discrepancies between the model and human that created the data necessitate non-realistic "residual" forces to keep the model from falling over when dynamically reproducing most movements. A 6 dof joint between the waist segment and the global frame generate the external forces. A weak, limited spring constrains the waist segment to achieve its recorded pose relative to the global frame. We also experimented with attaching the external constraint to the head segment or the feet with little noticeable difference. The non-realistic external forces (residuals) account for noise as well as discrepancies between the model and the human generating the data. In particular, differences in how the feet interact with the ground cause errors in our analysis. We found that, in most cases, it is only necessary to constrain two angular degrees of freedom (pitch and roll), leaving the other four external degrees of freedom un-powered. The two angular constraints keep the body from falling over but allow it to move about through simulated ground interactions. The simulation can reproduce highly dynamic motions (Figure 5.7).

If the joint springs use anything less than infinite stiffness, the joint forces will not be sufficient to exactly duplicate the input kinematic pose sequence. Although, without the marker constraints, the simulated model is somewhat less-likely

to be singular, it is still generally necessary to allow the constraints some slippage for stability. Although this slippage might make our method slightly less accurate than optimization-based methods, it has several advantages. The principal features of this method are that it is very fast, it is free, and it easily handles ground contact constraints, even under bipedal stance (closed kinematic loops).

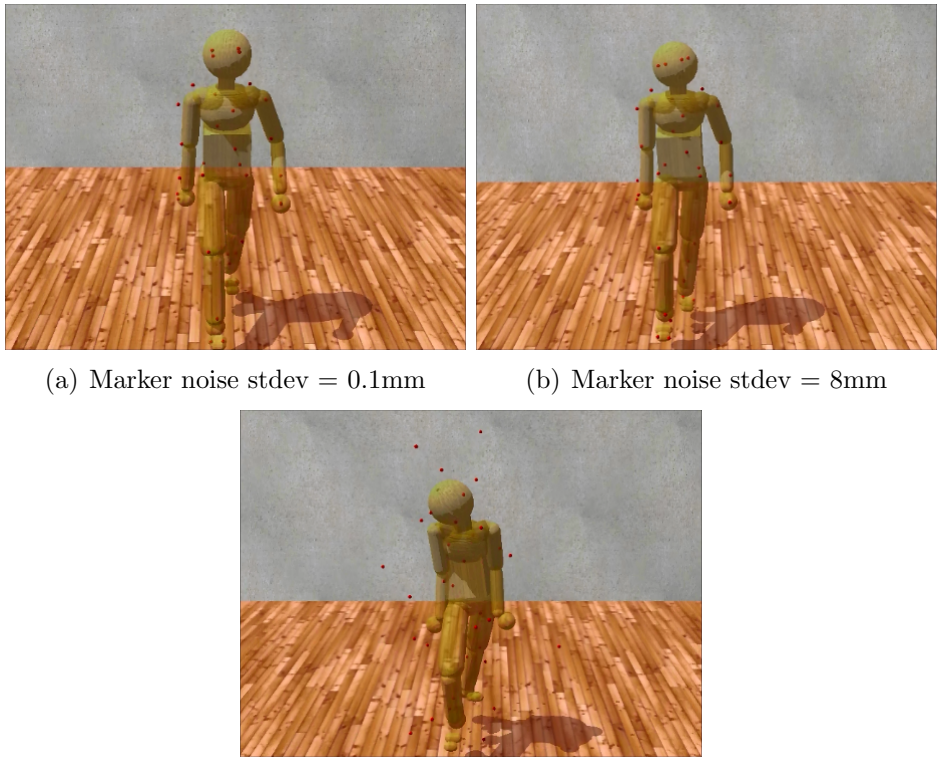
5.4 Experimental Results

Several experiments provide qualitative and quantitative validation of the physics-based movement analysis techniques described here. These experiments either use synthesized motion data as ground truth measurements or extra sensors for validating results.

5.4.1 Synthesized treadmill walking

Inverse dynamics computations rely on first finding the model's pose. We tested both steps by studying eight steps of marker data captured from treadmill walking. For this computation we used data sampled at 60 hz. The movement lasts a little longer than 4 seconds, giving us 260 frames of data. The aim of this study was to assess the effect of sensor noise on the results and compare the joint angles and torques found with our method to those used to generate marker data. We used an experimental process similar to that employed in [88].

We used a preliminary pass through the data to generate synthesized "ground truth" marker, pose, and torque data. After using the physics-based inverse kinematics to compute joint angles, we constrained the body to use forward dynamics



(a) Marker noise stdev = 0.1mm

(b) Marker noise stdev = 8mm

(c) Marker noise stdev = 64mm

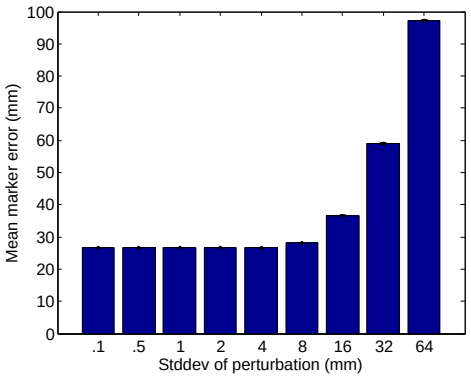
Figure 5.8: Poses generated by forward dynamics using forces obtained from inverse dynamics based on Gaussian perturbed walking data. Although at very high levels of noise, the model follows the reference motion poorly, the movement still looks, qualitatively, like walking.

to reproduce the joint angles with internal torques (and residual forces at the waist segment). As the model performed the movement, we recorded the global position of the marker attachment points. We also recorded the forces used and the resulting joint angles. Thus we had synthetic "ground truth" data directly from the model.

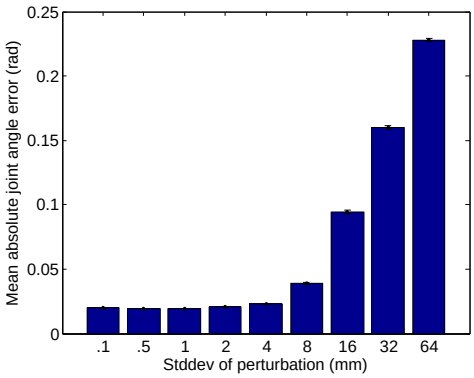
Using the synthetic marker data, we then analyzed the process described in this dissertation by perturbing all marker positions at each frame in time along all three axes with mean-centered Gaussian noise of a controlled standard deviation. Applying physics-based pose-fitting followed by inverse dynamics produced a new set of virtual marker positions, joint angles, and torques. We repeated the process twenty times for each noise-level at nine different standard-deviations. Standard-deviations, in mm, were (0.1, 0.5, 1, 2, 4, 8, 16, 32, 64). Example poses from the 0.1mm, 8mm, and 64mm noise levels are shown in Fig. 5.8.

Gaussian perturbations render the marker data dynamically inconsistent. This dynamic inconsistency also pushes a constrained system toward singularity, making it more difficult to solve numerically. We tested with very high levels of noise to see if they would slow the system down, or prevent it from finding any solution at all. In all cases, the system analyzed the perturbed data in real-time, finding pose data and dynamics data to fit the marker data.

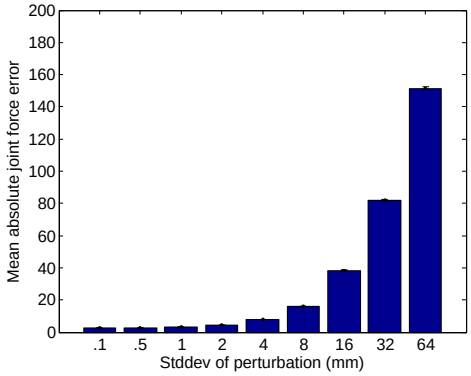
After running through an inverse kinematics pass, an inverse dynamics pass, and a forward dynamics pass for each trial run, we compared the marker attachment points, joint angles, and joint torques from the forward dynamics pass to the synthetic ground truth data. Figure 5.9 shows the mean error for across all degrees of freedom and frames of time for each quantity measured. Although the perturbations



(a) Marker error

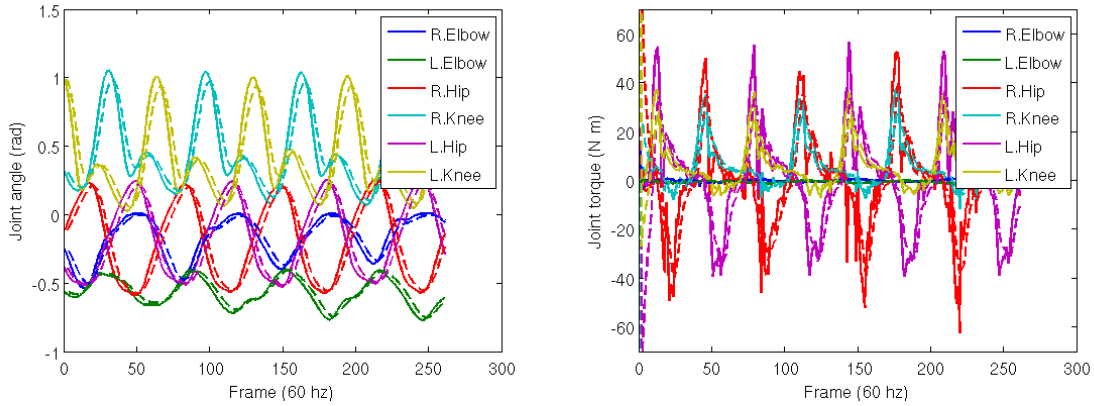


(b) Angle error



(c) Torque error

Figure 5.9: Error of marker attachment points, joint angles, and internal torques resulting from physics-based inverse kinematics and inverse dynamics used to analyze perturbed marker data. Error bars show standard error of the mean.



(a) Selected joint angles

(b) Selected joint torques

Figure 5.10: Trajectories of selected degrees of freedom from the perturbation study. Solid lines show ground truth. Dashed lines show computed data. Simulated spring forces make the computed data lag behind and smooth the ground truth.

make the marker data dynamically inconsistent, our results show that small amounts of noise have very little effect on the computed measurements. There is a systematic error in both the marker positions and joint angles caused by the fact that the constraints behave like springs. The spring-like behavior causes the marker positions and joint angles to lag behind their targets by a small amount and dampens the overall movement. This lag and damping are apparent in Fig. 5.10 comparing individual trajectories for selected dimensions of the joint angles and torques.

As shown in Fig. 5.10, the data follow ground truth very well under low noise conditions. It worth noting here that, for forward dynamics to work, we had to be very careful to exactly recreate the initial conditions from which the inverse dynamics were computed. Even the smallest amount of floating point variation would result in chaotic destabilization after half-a-second or less of forward dynamics. The inverted

pendulum nature of a humanoid character is inherently unstable. Small errors in body pose mean that the measured forces become increasingly wrong for the current conditions, pushing the body into strange places. This chaos comes as no surprise because it is well-known that completely open-loop control is very unstable.

5.4.2 Weighted and unweighted reaching with EMG

Physics-based inverse dynamics computes net-torque at the joint anchor. This computation cannot measure muscle co-contraction of opposing muscle pairs which gives stability to a movement under external perturbation. Using electromyogram sensors incorporated into the virtual reality environment (Chap. 3) to measure trapezius muscle activation, we compared physics-based torque measurements to integrated muscle activity. We placed EMG electrodes on the shoulder (trapezius) muscles to simultaneously measure muscle activity and motion capture during a reaching experiment. A subject reached to high and low virtual targets over multiple trials. The experiment measured the responses to both unweighted and weighted conditions. In the latter the subject had a four kilogram weight attached to the wrist of the reaching arm. The high correlation (r=0.88) suggests that the computed torques are directly related to actual muscle activations (Fig. 5.11).

5.4.3 Ground force comparison from standing data

Discrepancies between the model and reality make it so that the dynamic model falls over unless action is taken to stabilize it. Adjustments to internal joint torques can be used to stabilize the body, but cause the body pose to deviate from its

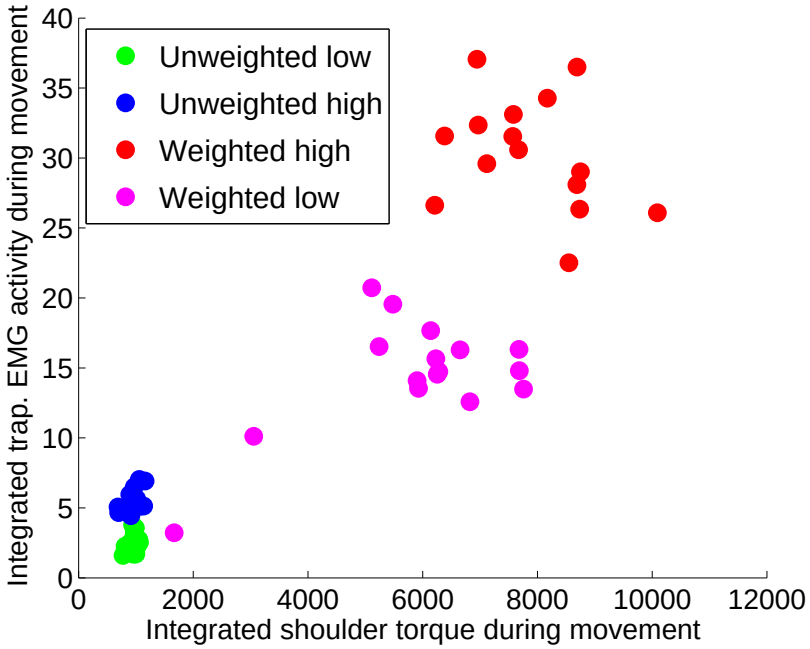


Figure 5.11: Subject reached forward to a 1.5m height or 2m height under unburdened conditions or wearing 4kg weight.

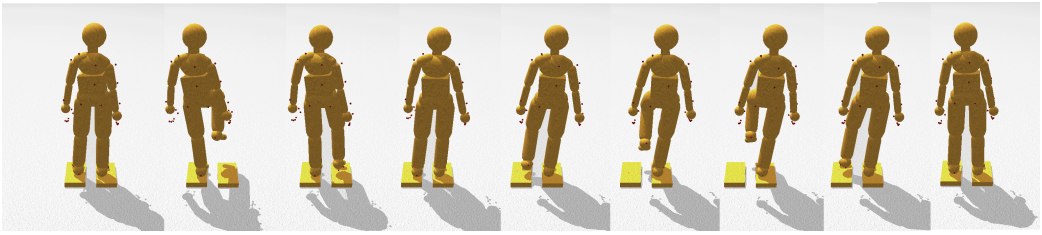


Figure 5.12: Using balance boards to measure ground forces, a subject transitions from balancing on one leg to balancing on the other.

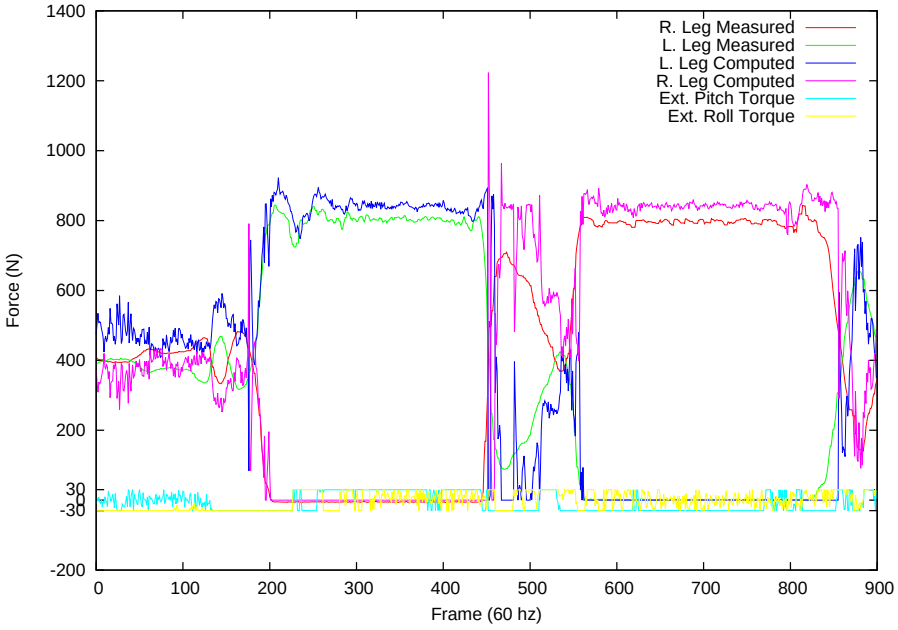


Figure 5.13: Ground reaction forces measured by Wii balance boards closely match those found by our inverse dynamics computations, even during bipedal stance. Residual torques, affecting only two rotational degrees of freedom (pitch and roll) are limited to 30 N m of torque, but are sufficient to maintain balance.

intended pose. In this work, we simply used non-realistic external forces. We attached a joint to the model's waist that would constrain it to reproduce the global orientations found during the pose-fitting pass. To minimize the effect of these external forces, we used torque limits on the amount of stabilizing torque available.

To test how much residual force torque is necessary relative to the internal joint torques, we obtained movement data together with ground force data from a pair of balance boards. For a simple movement such as transitioning from standing on one foot to the other (Fig. 5.12), residual angular torques of $30Nm$ were sufficient to keep the dynamic model quite close to its target trajectory. Figure 5.13 compares the sensor-measured ground forces for the right and left feet (red and green lines) to the computed ground forces found through physics-based inverse dynamics (blue and pink lines). Even during bipedal stance, the forces come surprisingly close. The largest discrepancies come during transition from one foot to the other. These discrepancies can be blamed largely on poor collision detection resulting from an abstract model of the foot.

5.5 Summary

With these tools provided by robust physics simulation, we can easily produce sequences of movement with accompanying torques and forces associated with a given set of conditions. Trajectory sequences, stored in a database, can be interactively modified and replayed from any point in time under different torque and body conditions. Although our method for computing inverse dynamics requires residual forces, a few changes to the physics code would allow us to account for these residual

forces with internal degrees of freedom as described in Chapter 4.

It should be possible to extend this method to use approximations of actual muscles instead of raw joint torques. Muscles and tendons are springy. A natural extension of this analysis work is to create constraints modeling individual muscles or muscle pairs attached at appropriate insertion points on the model's "bones". An important future step in movement analysis is to find relationships between human movements and the tasks they are executing. We suspect that movement trajectories found through our method can be sparsely fit with a dictionary of basis functions and that these dictionaries can be tuned for general-purpose motor control. Using the virtual environments and analysis techniques developed in this dissertation, future research will be able to explore these ideas and many more.

Chapter 6

Simulation-based Movement Synthesis

General purpose dynamic control of humanoid characters to achieve long term goals is complicated and beyond the scope of this work. We are able to synthesize some interesting movements on a frame-by-frame basis and adapt recorded movements to achieve new purposes using real-time, simulation-based techniques similar to those used for analysis in Chapter 5. This research uses constrained forward dynamics to solve inverse kinematics and manipulate the model end-effectors. Unlike computing the Jacobian in generalized coordinates, a physics-based approach makes it simple to apply multiple constraints to a model. It is also easy to apply partial constraints. For example, one might constrain an end effector to achieve a given height above the ground without constraining side-to-side movement at all. The general method presented in Chapter 5. This chapter discusses different ways we use the method create new movements.

Adapting a movement to meet a positional constraint can be as simple as creating a synthetic "marker" that then drags the model into place. It is also a simple matter to adapt data computed through movement analysis using one model to make a second model move. A limiting factor in using physical simulation as a tool for synthesizing movement is that it works best as a per-frame technique. We

must be able to specify a proper behavior for each computed moment of time. Even so, the physics engine can be a powerful tool for synthesizing movement.

6.1 Inverse Kinematics

In animation literature, inverse kinematics typically involves adapting a model pose so that an end effector satisfies some constraint. Just as when fitting a model to motion capture data, we attach a marker p_i with a ball-and-socket joint to some point on a corresponding body b_j. In this work, we assigned the point by hand and defined a trajectory for it to follow. To the physics engine, since p_i is kinematic, the joint essentially defines a constraint on b_i, stating that the point the marker is attached to must match the marker's velocity over time.

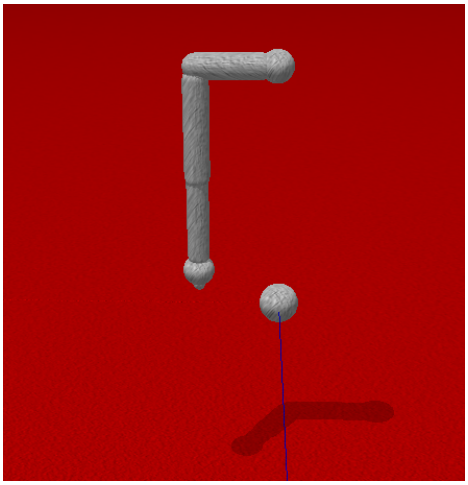


Figure 6.1: A simple, five-segment arm with eight degrees of freedom modeled using the Open Dynamics Engine physics simulation library (ODE).

We illustrate the physics-based method for synthesizing movement for an articulated body using a simplified arm model. Using a five-segment arm with four

universal joints (8 degrees of freedom, see Fig. 6.1), we first rigidly attach one end of the arm to the environment. We connect the other end of the model (the end-effector) to a "control marker" using a ball-and-socket joint. The control marker kinematically follows a cubic spline trajectory from its initial state to a target position and velocity, dragging the arm along behind it. The physics engine finds joint angles that satisfy the system of constraints as described in Chapter 4.

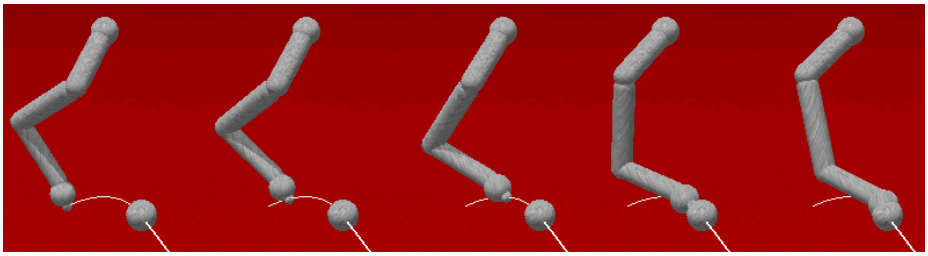


Figure 6.2: The control marker follows a cubic-spline trajectory lasting one second from its initial position and velocity to its target position and velocity, bringing the arm along with it. Time increases from left to right.

The spline path describes a trajectory lasting one second from start to finish. After the control marker reaches the target state, the arm joints return the arm near to its starting state. Time restarts and the arm reproduces its original trajectory using constraints on the internal joint angles. Finally, the arm returns to the start state and replays the torques computed during the second pass. During each pass we recorded the joint angles and end-point position for the arm (Fig. 6.3).

The joint angles and end-point position very closely matched the original trajectory. Figure 6.3 shows that the replayed torque sequence also closely tracks the original values. This method is prone to break down if the spline trajectory forces the joint angles against their limits or beyond the reach of the articulated charac-

ter. Higher-level planning, a subject for future work, is necessary to reach around singularities. This fact highlights an advantage of using human data to synthesize movement. The human solves these harder problems. Using our method, adding additional constraints to movements described by human behavior data is relatively painless.

The simulation finds a body pose that does a good job satisfying the marker constraints. Once we have that body pose, we can use it as a new resting pose and modify the movement using the same technique. If we want the hand to reach a little farther, we can constrain it to do so and the physics engine finds a solution. It is important, however, to balance the springs. Working with a full humanoid model, if we try to move the hand but the arm joints are too stiff, we might end up dragging the entire body instead of just extending our reach. The nice thing is that this method provides an intuitive way to control the result that you will get. Weakening the springs or modifying the setpoints controlling the arms biases the movement to selected joints. It is also simple to add additional constraints to keep the waist in place or the feet planted. An underactuated model will require additional constraints or planning to ensure that the movements generated through physics-based inverse kinematics satisfy balance constraints and other criteria necessary to keep the model from falling over. The usefulness of this approach is that it works quickly and integrates seamlessly into systems already using ODE for forward dynamics. Adding an arbitrary number of constraints to a character model makes it possible for an animator or robot operator to achieve desired outcomes to make an animation look good or to find a model pose sequence that places an end-effector where it is needed.

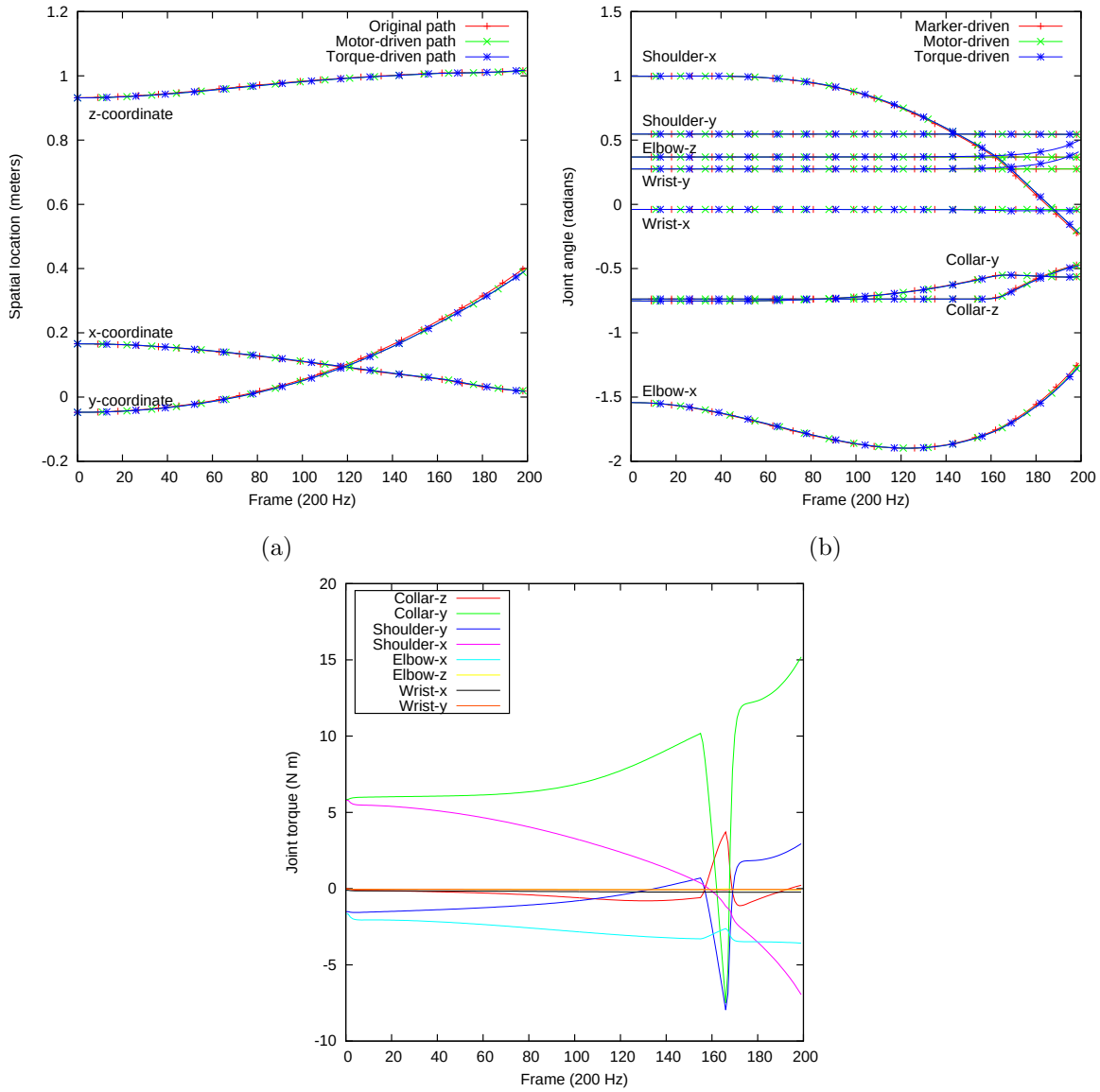


Figure 6.3: (a) *Endpoint trajectory* and (b) *joint angles* of the arm are almost identical through the duration of the movement for all three control cases: kinematic, built-in joint motors, and replayed torques. (c) The *joint torques* used to drive the disembodied arm along the spline trajectory show a discontinuity that occurs around the point where the shoulder passes near its joint limit when the shoulder rolls back slightly during the reaching movement.

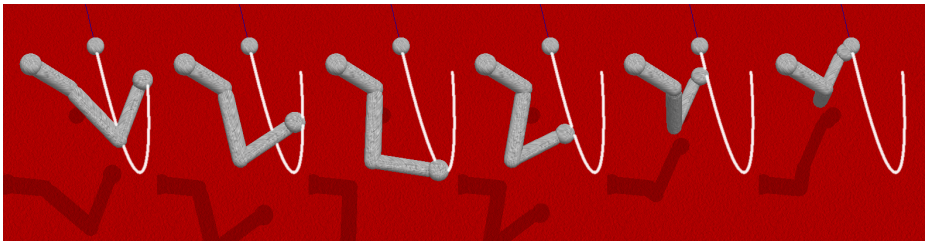


Figure 6.4: The endpoint of a four-joint arm (8 degrees of freedom) with slightly stiff joints is forced to follow a spline trajectory. The physics engine performs the inverse kinematics without any additional programming effort.

6.2 Retargeting

The method works well, in real-time, even when the simulated model dimensions and mass are changed drastically, allowing us to quickly retarget the markers to another body model. Simply changing the model without any other changes will not do great retargeting, of course. Unfortunately, there cannot exist a perfect way to automatically accomplish high-quality, general-purpose retargeting without a firm understanding of the purpose of the movement. If the movement is clapping hands, you need to constrain the hands to meet at the right time. If the movement is locomotion, you need to constrain the limbs to produce the appropriate ground forces at the right times and you need to decide if a larger body should walk faster because of its longer legs or if it should use a short stride to match the original movement. These decisions cannot, in general, be made automatically because there are situations where an animator might want either. It falls on the animator or operator to determine which constraints, internal joint angle control or exocentric target-based control, are most appropriate to satisfy higher-level constraints such as traversing a sequence of stepping stones. Interesting work toward capturing what constitutes an

"action" is available in [52]. Future work on retargeting can build on this work to represent "actions" as constraints on the global and relative positions and orientations of body segments and then take advantage of our physics-based approach to synthesize new movements directly with the physics engine.

6.3 Abstract Models

To synthesize dynamic movements with the physics engine we employ abstract models that describe movement trajectories for key model elements. One strategy for controlling a humanoid agent is to first develop a control strategy for a low-dimensional system and map the controls onto a higher dimensional system. The spring-loaded inverted pendulum is one such model for locomotion [74, 85].

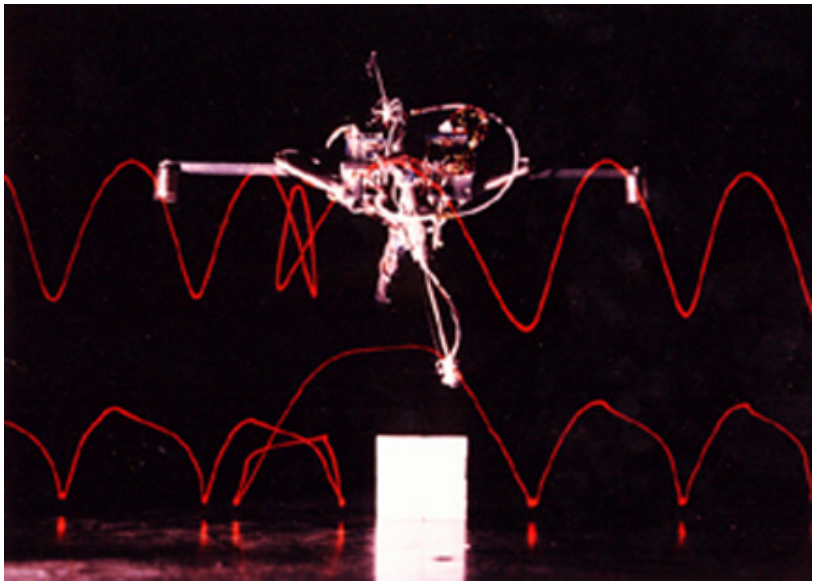


Figure 6.5: Raibert's 2d hopper[85], essentially a massive cross-piece hinged on a piston-driven spring. The machine is rigidly tethered to a center pole to constrain its movement to two dimensions.

The purpose of an abstract model is to provide a low dimensional control space that admits an effective control strategy. Given a control strategy for driving a simple physical model, the physics engine can be employed to project that strategy onto a more complex model. For example, we reproduced Raibert's 2d hopping model[85] (see Figure 6.5). This design is based on a spring-loaded inverted pendulum. The controller functions according to two discrete states: while the foot is on the ground, it attempts to stabilize the "torso" segment and control angular momentum. While the hopper is in the air, it attempts to place the foot (based on forward velocity) so that the center of mass spends an equal amount of time on either side of the foot-plant. This model assumes that the mass of the leg is trivial in comparison to the body. Although the assumption is not quite true, the feedback loops controlling the system are sufficiently forgiving that it still works very well. The system is simple, well-described in Raibert's book[85], and easy to reproduce (Figure 6.6). This basic model can serve as an abstraction for where to place feet while walking or jumping to maintain stability.

Mapping the hopper's controller onto a humanoid is simple using our physics-based technique. The hopper's center of mass and orientation provide a constraint on the model's upper body. The hopper's foot serves as a constraint on the humanoid's feet. The physics engine then solves, first the kinematics for a hopping humanoid and then the dynamics (Fig. 6.6). The physics engine software makes this approach easy to implement.

The hopper abstraction cannot stop hopping or it will fall over. It requires a ballistic phase in order to place the foot and correct gradual tipping that invariably oc-

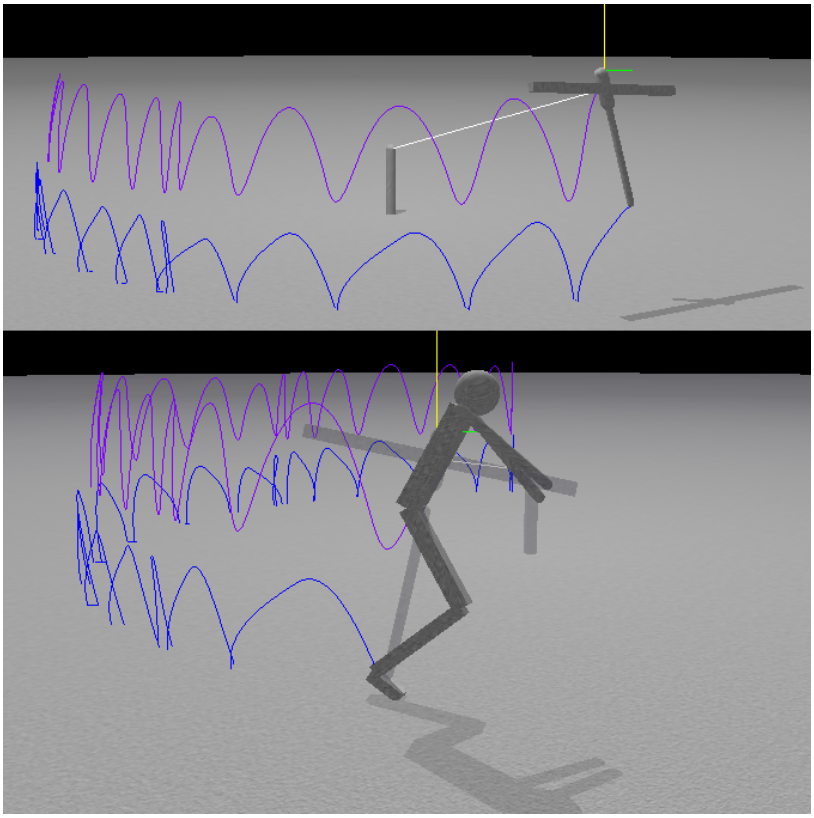


Figure 6.6: *Top*: Our reproduction of Raibert's 2d hopper. *Bottom*: The hopper model can be an abstract dynamic model for a hopping human.

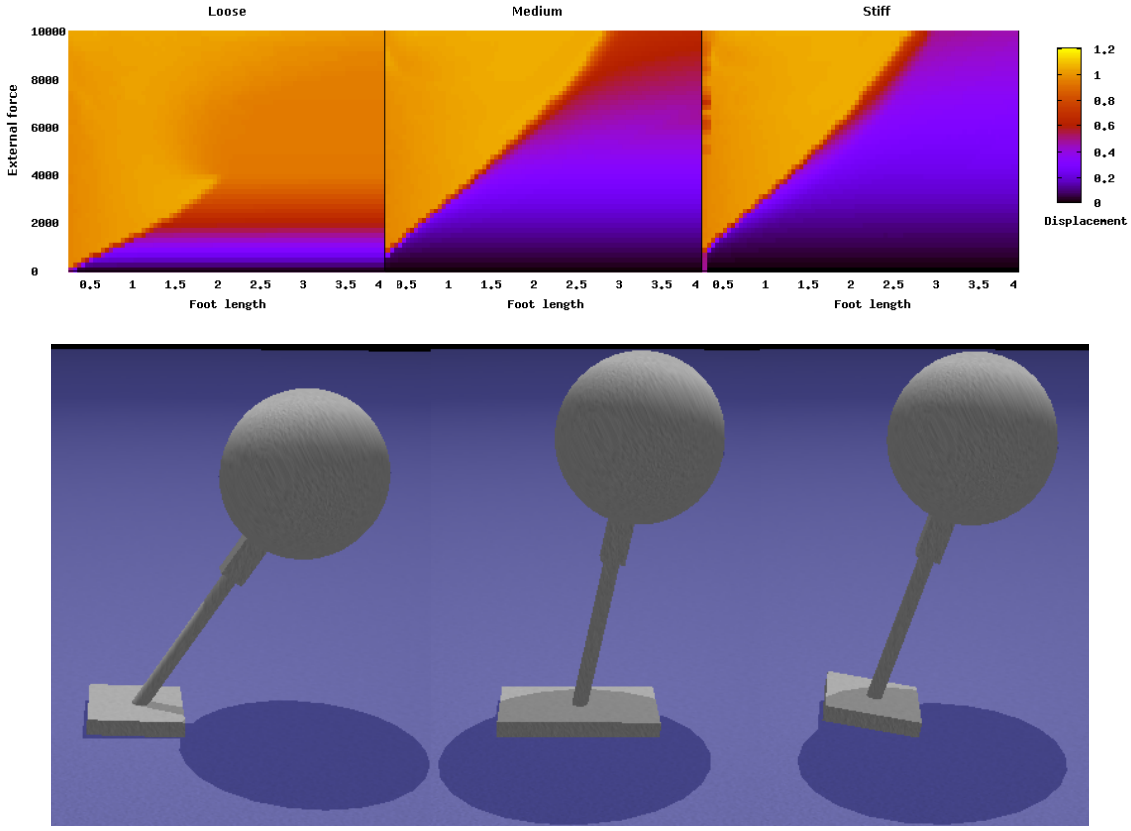
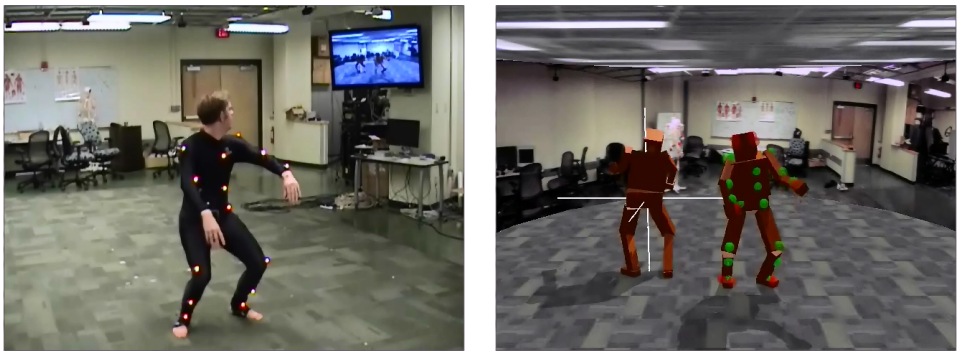


Figure 6.7: *Top:* Three plots showing the effect of stiffness and foot size on stability of an inverted pendulum with a foot. A larger base and a stiffer ankle resist tipping. *Bottom:* Abstraction with foot: Loose ankle, Big foot, Stiff ankle.

curs. Although the hopper is never instantaneously stable, the sequence of foot-plants it produces creates an abstract base with the center of mass stably centered when observed at the right temporal scope. We added a simulated foot to the hopper model and experimented with the effect of ankle size and stiffness on stability (Figure 6.7). This experiment showed that it is necessary to keep the ankle moderately stiff while on the ground to avoid falling over. When the system is perturbed in any direction, the center of pressure naturally shifts in the same direction to counteract the force and restore stability. Higher level control abstractions can rely on this phenomenon and begin hopping when perturbations exceed the lower level's tolerance. Building up from this abstract model is interesting future work. Using the current system, human data can provide necessary trajectories to synthesize interesting movements such as tele-operating a synthesized robot.

The processes of fitting the model pose to marker data and solving the inverse dynamics with internal forces both run in real-time. Using this process we were able to create an interface that allowed a human subject to control a simulated humanoid robot (Figure 6.8). Tele-operating the simulated robot employed two humanoid models. One model was dragged along by marker data, producing a kinematic pose. The second model, was constrained to use internal forces to copy the pose of the kinematic model without any residual forces. In approximately a minute, without any previous experience, the human was able to control the dynamic model to stand up straight and balance. Although the "operator" was the creator of the programming mapping the motion capture markers to kinematic and dynamic models, no particular insight was necessary to control the dynamic agent. Programming this behavior by hand



(a) A human subject monitors a kinematic model and a dynamic model following his movements in real-time.

(b) The kinematic model (right) is constrained to duplicate the human pose in space. The dynamic model (left) uses forces to reproduce the kinematic pose, but is free to fall over.

Figure 6.8: Our technique for finding the model pose and computing inverse kinematics runs in real-time. Here, a human subject wearing the motion capture suit is able, after one practice trial, to move his body so that the dynamic model (without any external forces) goes from lying to stable standing.

would be challenging because of the many degrees of freedom and the inherent instability of the inverted-pendulum. Working from a human model and taking advantage of human feedback made it relatively simple.

6.4 Summary

We have described how our physics-based technique makes it possible to synthesize some novel movements. The synthesized movements are largely kinematic, rather than dynamic, but they are still effective and can be part of a dynamic simulated environment, interacting with other simulated objects for games or other purposes.

Combing these techniques with learning approaches to create intelligent con-

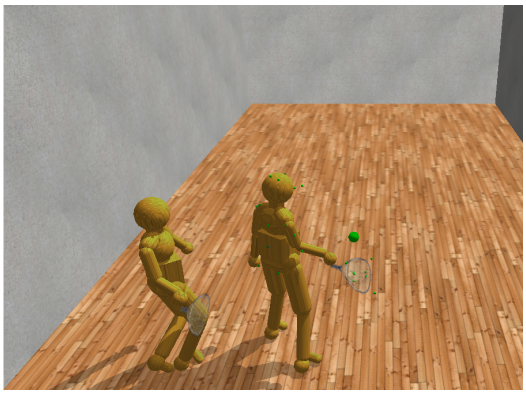


Figure 6.9: A virtual racquetball opponent could learn appropriate motor behaviors interactively from a human playing in virtual reality.

trollers presents exciting possibilities. For example, a human engaged in some virtual activity automatically provides training data that can be used to improve a virtual agent doing the same activity (Fig. 6.9).

Another interesting way to use physics-based movement synthesis is in creating an impedance controller for a robotic system. Impedance control involves making a system respond to forces as though it had different dynamic properties than its actual mechanics[47]; e.g., a robot arm might act like it weighs less than it actually does by actively moving in the direction of an externally applied force. Similar to the use of abstract models to make the control problem tractable, it should be possible to simultaneously simulate a robot model with the desired impedance and another model with the robot's actual impedance. Wrapping these models into a feedback loop with the actual robot controller and sensors might facilitate implementation of impedance control.

Chapter 7

Conclusion and Future Work

The problems addressed and artifacts produced throughout the course of this research are useful and interesting to the field of computer science and potentially to other disciplines. The techniques discussed in this document address several notable problems in capturing, modeling, and synthesizing human movement with easily applied techniques that leverage robust, freely-available simulation code. We have shown how physical simulation software, already needed for forward dynamics computations in a game or experimental virtual environment, can be used to accomplish analysis and synthesis of movement data at interactive rates. Robust dynamics code, made freely available through ODE, is easily adapted into a robust controller with intuitive parameters for driving a simulated model toward a target state. Although the analysis may not be as accurate as needed for some clinical biomechanics applications and we have not had an opportunity to validate the use of our approach for controlling genuine, physical robots, the analysis and synthesis techniques are sufficiently fast and accurate to be used in interactive applications such as games and experiments in virtual reality.

7.1 Summary of Contributions

The principal contribution of this research has been to demonstrate how the constraint solver algorithms used in general-purpose physics simulation code, such as the Open Dynamics Engine, can be readily adapted to accomplish many useful tasks besides simulating constrained forward dynamics. These tasks include several steps in using motion capture data to analyze and synthesize humanoid movements:

1. Fitting a character model to motion-capture data by finding joint pivot positions and marker attachment points.

2. Modeling marker trajectories as infinite point masses and attaching them to a character model in a way that constrains the model to reproduce a captured pose.

3. Computing inverse dynamics forces to reproduce captured movements in a simulated model and thus gain insight into the effort involved in a specific human task.

4. Modifying forces and movements for a new model or related goal.

Used for these purposes, the ODE simulation software becomes a fast, robust, intuitive, and inexpensive multi-purpose tool for simulating, analyzing, and synthesizing movement.

The mathematical analysis in this document provides useful insights into what makes these different tasks possible. In addition to our software interface for interactively analyzing movement data using the techniques presented here, this research

has produced multiple software contributions for the simulation library used in this research. Among these software contributions are code patches for increasing stability and for providing useful general-purpose constraints. These improvements have been submitted to the ODE open-source code-repository. Some of our contributions have already been incorporated into the public project and others will be soon. Because ODE is widely used for research as well as for physics-based animation in games, these contributions stand to benefit a large population.

Finally, the virtual reality environment code and interactive interface for applying the above tools have already proven useful for many purposes. Young people visiting the lab through university sponsored events have expressed their excitement for pursuing this type of research after having a chance to experience the interactive virtual environment. Multiple psychophysical experiments using the system are producing insights into human attention and behavior. The example implementation produced as part of this research has already been used for research measuring human effort and will be instrumental in supporting future work in better understanding the computation underlying human movement.

7.2 Future Work

The research involved in this dissertation covers a lot of material, crossing traditional discipline boundaries. There are many interesting avenues available for further exploration. Some have come up throughout the course of this document. We discuss just a few here. These future research efforts relate to improving simulation/dynamics algorithms, expanding the usefulness of or virtual environment,

employing the methods described her for additional purposes, and building more advanced techniques.

The contributions we have made to the simulation platform may be useful in many ways. General purpose joints may be used to help create new dynamic systems with interesting properties. A proper exploration of constraint theory and the relationship between different types of joints seems interesting.

Stability in simulation remains an issue. We have made some contributions toward making the simulation more accurate without excessively affecting computational complexity. However, in some cases, a model will still fly apart. This phenomenon needs to be explored systematically to discover exactly what conditions lead to instabilities so that these can be addressed if possible. One particular open question is the problem of dealing with disparate mass ratios between bodies that share a constraint. Large mass-ratios are a problem for multiple reasons. When the masses are inverted and added together, the lighter-weight body dominates the equation. High mass and low mass bodies in the same constraint also tend to make the constraint matrix ill-conditioned. These problems may be addressable with sufficient insight into how the constraint solver and discrete mathematics work. Our derivation (Chap. 4) of the constraint system suggests that it might be possible to condition the effective inverse-mass matrix by modifying the constraint Jacobian in a way that is equivalent to scaling the distances used to represent the positions of the rigid bodies.

Although we focused primarily on building tools with and expanding the capabilities of the dynamic simulation code, the virtual reality technology was also fun to work with. The potential applications and extensions of this work are limitless.

Research opportunities using VR for artistic installations, entertainment, and education abound. Many questions regarding human movement and behavior can also be readily addressed by systematically controlling conditions in virtual reality. We are in the process of designing a handful of experiments to explore the characteristics of preplanned movements in comparison to planned movements. The ability to precisely control information presented to human subjects while still presenting a naturalistic environment promises to support significant research into the computations underlying the human brain. Conveniently, the analysis methods presented in Chapter 5 can be applied directly to kinematically controlling an avatar that allows research subjects to see themselves within the virtual environment and interact with other objects in the world when appropriate. Furthermore, the inverse dynamics methods can then be used to rapidly estimate the effort exerted during the experiment, allowing torque-dependent stimuli.

Because the inverse kinematics and inverse dynamics methods presented here run in real-time and integrate seamlessly with physical simulation, it is possible to make a physical model dynamically imitate a human's behavior at interactive speeds. A human wearing motion-capture equipment can then 'teleoperate' a model while responding appropriately to the conditions it faces. With a sufficiently accurate model, this approach could also be applied to real robots, similar to the work done by [94], but controlling an entire body at once. A human-robot-interaction paradigm of this sort has the potential to make robot control very intuitive. A direct mapping may mean that robot operators can easily learn the consequences of their own movements on the robot system and thus rapidly adapt their human feedback loops to control

the machine.

The techniques presented in this dissertation work with kinematics and dynamics primarily on a per-frame basis, limiting what they can do. Hierarchical methods show promise for both analyzing and controlling humanoid movement. It would be interesting to build a hierarchical dynamics system such that detail movements happen on a local basis over short periods of time but are abstracted away into coarser elements over larger periods of time. Some work has already been done exploring basis functions as representations of movement and general human behaviors. A general purpose controller for a humanoid robot will need the ability to encapsulate behaviors as hierarchies of feedback loops. One particular avenue worth additional attention is the problem of fitting movement data with parametrized functions or distributions that would enable the production of appropriate movements under novel conditions.

For biomechanics research, it can be important to look at the effects of individual muscles instead of abstracting them away into aggregate joint torques. We attach markers to a simulated humanoid model and drag it into position with spring-like constraints when computing model poise from motion capture data. Future efforts may directly model muscles with constraints instead of abstracting muscle activity into joint torques.

Movement analysis and synthesis are exciting areas to be working in. The research is multi-disciplinary by nature. There are potential applications that can improve quality of life in many different ways. We hope this research will help animate characters and robots for entertaining, training, and working to benefit society. We hope it will be useful for diagnosing motor pathologies, assessing muscle injuries, or

simply working to better understand human movement.

Bibliography

[1] Pieter abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *Int. J. Rob. Res.*, 29:1608–1639, November 2010.

[2] Brian F. Allen and Petros Faloutsos. Evolved controllers for simulated locomotion. In *Proceedings of the 2nd International Workshop on Motion in Games*, MIG '09, pages 219–230, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] Michael S Andersen, Daniel L Benoit, Michael Damsgaard, Dan K Ramsey, and John Rasmussen. Do kinematic models reduce the effects of soft tissue artefacts in skin marker-based motion analysis? an in vivo study of knee kinematics. *Journal of Biomechanics*, 43(2):268–273, 2010.

[4] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[5] Okan Arikan. Compression of motion capture databases. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 890–897, New York, NY, USA, 2006. ACM.

[6] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21:483–490, July 2002.

[7] Okan Arikan, David A. Forsyth, and James F. O'Brien. Pushing people around. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 59–66, New York, NY, USA, 2005. ACM.

[8] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 23–34, New York, NY, USA, 1994. ACM.

[9] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *SIGGRAPH Comput. Graph.*, 22(4):179–188, June 1988.

[10] Jan Baumann, Bjorn Kruger, Arno Zinke, and Andreas Weber. Data-driven completion of motion capture data. In *Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)*, 2011.

[11] J Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

[12] Nicholai Bernstein. The co-ordination and regulation of movement. *London: Pergamon*, 1967.

[13] Rodney Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, mar 1986.

[14] Grigore Burdea and Philippe Coiffet. Virtual reality technology. *Presence: Teleoperators and virtual environments*, 12(6):663–664, 2003.

[15] Samuel R Buss. Accurate and efficient simulation of rigid-body rotations. *Journal of Computational Physics*, 164(2):377–406, 2000.

[16] Erin Catto. Iterative dynamics with temporal coherence. Technical report, Crystal Dynamics, Menlo Park, California, 2005.

[17] Erin Catto. Soft constraints: Reinventing the spring. `http://box2d.googlecode.com/files/GDC2011_Catto_Erin_Soft_Constraints.pdf`, 2011. Game Developers Conference (GDC).

[18] Ross A Clark, Adam L Bryant, Yonghao Pua, Paul McCrory, Kim Bennell, and Michael Hunt. Validity and reliability of the nintendo wii balance board for assessment of standing balance. *Gait & posture*, 31(3):307–310, 2010.

[19] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

[20] Perry R. Cook and Gary P. Scavone. The synthesis toolkit (stk). International Computer Music Conference, Beijing, China, 1999.

[21] Joseph L Cooper and Dana Ballard. Realtime, physics-based marker following. In *Motion in Games*, pages 350–361. Springer, 2012.

[22] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 130:1–130:9, New York, NY, USA, 2010. ACM.

[23] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Trans. Graph.*, 30(4):59:1–59:12, August 2011.

[24] Luis Cruz, Luis Paulo Reis, Nuno Lau, and Armando Sousa. Optimization approach for the development of humanoid robots behaviors. In *Advances in Artificial Intelligence–IBERAMIA 2012*, pages 491–500. Springer, 2012.

[25] Maria Cutumisu and Duane Szafron. An architecture for game behavior AI: Behavior multi-queues. In *Artificial Intelligence and Interactive Digital Entertainment*, pages 20–27, 2009.

[26] Edilson De Aguiar, Christian Theobalt, and Hans-Peter Seidel. Automatic learning of articulated skeletons from 3d marker trajectories. In *Advances in Visual Computing*, pages 485–494. Springer, 2006.

[27] Edilson de Aguiar, Christian Theobalt, and Hans-Peter Seidel. Automatic learning of articulated skeletons from 3d marker trajectories. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Ara V. Nefian, Meenakshisundaram Gopi, Valerio Pascucci, Jiri Zara, Jose Molineros, Holger Theisel, and Thomas Malzbender, editors, *Advances in Visual Computing, Second International Symposium, ISVC 2006, Lake Tahoe, NV, USA, November 6-8, 2006 Proceedings, Part I*, volume 4291 of *Lecture Notes in Computer Science*, pages 485–494. Springer, 2006.

[28] Martin de Lasa and Aaron Hertzmann. Prioritized optimization for task-space

control. In *Intelligent Robots and Systems (IROS). IEEE/RSJ International Conference on*, pages 5755–5762, oct. 2009.

[29] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Trans. Graph.*, 29(4):131:1–131:10, July 2010.

[30] Emel Demircan, Luis Sentis, Vincent Sapio, and Oussama Khatib. Human motion reconstruction by direct control of marker trajectories. In Jadran Lenar?i? and Philippe Wenger, editors, *Advances in Robot Kinematics: Analysis and Design*, pages 263–272. Springer Netherlands, 2008.

[31] Gabriel Diaz, Joseph Cooper, Dmitry Kit, and Mary Hayhoe. Real-time recording and classification of eye-movements in an immersive virtual environment. *Journal of vision*, Accepted, 2013.

[32] Gabriel Diaz, Joseph Cooper, Constantin Rothkopf, and Mary Hayhoe. Internal models for predictive saccades in a natural interception task. *Journal of Vision*, 12(9):606–606, 2012.

[33] Gabriel Diaz, Joseph Cooper, Constantin Rothkopf, and Mary Hayhoe. Saccades to future ball location reveal memory-based prediction in a virtual-reality interception task. *Journal of vision*, 13(1), 2013.

[34] Open Dynamics Engine. ODE User Manual. `http://ode-wiki.org/wiki/`, 2013. [Online; accessed 19-July-2013].

[35] Kutluhan Erol, James Hendler, and Dana S. Nau. Htn planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on*

Artificial Intelligence (AAAI-94), volume 12, pages 1123–1128. AAAI Press, 1994.

[36] Roy Featherstone. *Rigid body dynamics algorithms*, volume 49. Springer Berlin, 2008.

[37] S. J. Fellows and A. F. Thilmann. The role of joint biomechanics in determining stretch reflex latency at the normal human ankle. *Experimental Brain Research*, 77:135–139, 1989. 10.1007/BF00250575.

[38] Thomas Geijtenbeek, N. Pronost, Arjan Egges, and M. H. Overmars. Interactive character animation using simulated physics. In *Eurographics (State of the Art Reports)*, 2011.

[39] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2):193–203, 1988.

[40] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[41] Teodor P Grantcharov, Linda Bardram, Peter Funch-Jensen, and Jacob Rosenberg. Learning curves and impact of previous operative experience on performance on a virtual reality simulator to test laparoscopic surgical skills. *The American journal of surgery*, 185(2):146–149, 2003.

[42] F Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3):29–48, 1998.

[43] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. *ACM Trans. Graph.*, 23(3):522–531, August 2004.

[44] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer, 2006.

[45] Hermann Haken. Synergetics of brain function. *International Journal of Psychophysiology*, 60(2):110 – 124, 2006. ¡ce:title¿Models and Theories of Brain Function with Special Emphasis on Cognitive Processing¡/ce:title¿.

[46] Rachel Heck and Michael Gleicher. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 129–136, New York, NY, USA, 2007. ACM.

[47] Neville Hogan. Impedance control: An approach to manipulation: Part illapplications. *Journal of dynamic systems, measurement, and control*, 107(2):17, 1985.

[48] Samuel Hornus, Jared Hoberock, Sylvain Lefebvre, and John Hart. Zp+: correct z-pass stencil shadows. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 195–202, New York, NY, USA, 2005. ACM.

[49] Leslie Ikemoto, Okan Arikan, and David Forsyth. Knowing when to put your foot down. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, I3D '06, pages 49–53, New York, NY, USA, 2006. ACM.

[50] Damian Isla. Handling complexity in the Halo 2 AI. *Game Developers Conference*, page 12, 2005.

[51] Sumit Jain and C. Karen Liu. Modal-space control for articulated characters. *ACM Trans. Graph.*, 30(5):118:1–118:12, October 2011.

[52] Bernhard Jung, HeniBen Amor, Guido Heumer, and Arnd Vitzthum. Action capture: A vr-based method for character animation. In Guido Brunnett, Sabine Coquillart, and Greg Welch, editors, *Virtual Realities*, pages 97–122. Springer Vienna, 2011.

[53] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1620–1626. IEEE, 2003.

[54] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 239–246. IEEE, 2001.

[55] Amir Karniel. Open questions in computational motor control. *J Integr Neurosci*, 10(3):385–411, Sep 2011.

[56] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation, IEEE Journal of*, 3(1):43–53, february 1987.

[57] O. Khatib, E. Demircan, V. DeSapio, L. Sentis, T. Besier, and S. Delp. Robotics-based synthesis of human motion. *Journal of Physiology - Paris*, 103(3-5):211–219, 2009.

[58] Adam G Kirk, James F O'Brien, and David A Forsyth. Skeletal parameter estimation from optical motion capture data. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 782–788. IEEE, 2005.

[59] Evangelos Kokkevis. Practical physics for articulated characters. In *Game Developers Conference (GDC)*, 2004.

[60] Eugenia M Kolasinski. Simulator sickness in virtual environments. Technical report, DTIC Document, 1995.

[61] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, August 2004.

[62] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21:473–482, July 2002.

[63] P.G. Kry, L. Reveret, F. Faure, and M.-P. Cani. Modal locomotion: Animating virtual characters with natural vibrations. *Computer Graphics Forum*, 28(2):289–298, 2009.

[64] Arthor D Kuo. A least-squares estimation approach to improving the precision of inverse dynamics computations. *Transactions-American Society Of Mechanical Engineers Journal Of Biomechanical Engineering*, 120:148–159, 1998.

[65] Cornelius Lanczos. *The variational principles of mechanics*. Number 4. Courier Dover Publications, 1970.

[66] Michael S Landy, Laurence T Maloney, Elizabeth B Johnston, and Mark Young. Measurement and modeling of depth cue combination: In defense of weak fusion. *Vision research*, 35(3):389–412, 1995.

[67] Mark L. Latash. *Neurophysiological Basis of Movement*. Human Kinetics Publishers, second edition, 2008.

[68] Mark L. Latash. *Synergy*. Oxford University Press, USA, 2008.

[69] Alberto Leardini, Lorenzo Chiari, Ugo Della Croce, and Aurelio Cappozzo. Human movement analysis using stereophotogrammetry: Part 3. soft tissue artifact assessment and compensation. *Gait & posture*, 21(2):212–225, 2005.

[70] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1071–1081, New York, NY, USA, 2005. ACM.

[71] C. Karen Liu and Sumit Jain. A quick tutorial on multibody dynamics. Technical report, Georgia Institute of Technology, 2012.

[72] Patrick MacAlpine, Daniel Urieli, Samuel Barrett, Shivaram Kalyanakrishnan, Francisco Barrera, Adrian Lopez-Mobilia, Nicolae Ştiurcă, Victor Vu, and Peter Stone. UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.

[73] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki. A learning approach to robotic table tennis. *IEEE Transactions on Robotics*, 21(4):767 – 771, Aug. 2005.

[74] Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.*, 29(4):71:1–71:8, July 2010.

[75] Karl M. Newell and David E. Vaillancourt. Dimensional change in motor learning. *Human Movement Science*, 20(4–5):695 – 715, 2001.

[76] Oliver Obst and Markus Rollmann. Spark-a generic simulator for physical multi-agent simulations. *Computer Systems Science and Engineering*, 20(5):347, 2005.

[77] US Government Federal Business Opportunities. Broad agency announcement DARPA robotics challenge tactical technology office (TTO) DARPA-BAA-12-39. Technical report, DARPA, April 2012.

[78] R. Parasuraman, T.B. Sheridan, and C.D. Wickens. A model for types and levels of human interaction with automation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(3):286–297, may 2000.

[79] Chunk-Kang Peng, Jeffrey M. Hausdorff, and Ary L. Goldberger. Fractal mechanisms in neural control: Human heartbeat and gait dynamics in health and disease. *Self-Organized Biological Dynamics and Nonlinear Control*, pages 66–96, 2000.

[80] Jan Peters, Katharina Mülling, and Jens Kober. Experiments with motor primitives to learn table tennis. In O. Khatib, V. Kumar, and G. Sukhatme, editors, *Experimental Robotics*, pages 1–13, Berlin, Germany, March 2010. Springer.

[81] Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami. Capture point: A step toward humanoid push recovery. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 200–207. IEEE, 2006.

[82] Dennis R Proffitt. Embodied perception and the economy of action. *Perspectives on psychological science*, 1(2):110–122, 2006.

[83] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, number 3.2, 2009.

[84] Rafidah Abd Rahim, Norhaida Mohd Suaib, and Abdullah Bade. Motion graph for character animation: Design considerations. In *Proceedings of the 2009 International Conference on Computer Technology and Development - Volume*

02, ICCTD '09, pages 435–439, Washington, DC, USA, 2009. IEEE Computer Society.

[85] Marc H. Raibert. *Legged robots that balance*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1986.

[86] Jeffrey A Reinbolt, Ajay Seth, and Scott L Delp. Simulation of human movement: applications using opensim. *Procedia IUTAM*, 2:186–198, 2011.

[87] P. Reist and R. Tedrake. Simulation-based LQR-trees with input and state constraints. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5504 –5510, may 2010.

[88] Christian D Remy and Darryl G Thelen. Optimal estimation of dynamically consistent kinematics and kinetics for forward dynamic simulation of gait. *Journal of biomechanical engineering*, 131(3):031005, 2009.

[89] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 514–521, New York, NY, USA, 2004. ACM.

[90] Stefan Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems*, volume 9, pages 1040–1046, Cambridge, MA, 1997. MIT Press.

[91] J. P. Scholz and Gregor Schner. The uncontrolled manifold concept: identifying control variables for a functional task. *Experimental Brain Research*, 126(3):289–306, 1999.

[92] L. Sentis, Jaeheung Park, and O. Khatib. Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *Robotics, IEEE Transactions on*, 26(3):483–501, june 2010.

[93] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(4):505–518, 2005.

[94] Adam Setapen, Michael Quinlan, and Peter Stone. Marionet: Motion acquisition for robots through iterative online evaluative training. In *Ninth International Conference on Autonomous Agents and Multiagent Systems - Agents Learning Interactively from Human Teachers Workshop (AAMAS - ALIHT)*, May 2010.

[95] Reza Shadmehr and Steven P. Wise. *Computational Neurobiology of Reaching and Pointing: A Foundation for Motor Learning*. MIT Press, Cambridge MA, 2005.

[96] Marius-CĂlin Silaghi, Ralf Plänkers, Ronan Boulic, Pascal Fua, and Daniel Thalmann. Local and global skeleton fitting techniques for optical motion capture. In *Modelling and Motion Capture Techniques for Virtual Environments*, pages 26–40. Springer, 1998.

[97] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 15–22, New York, NY, USA, 1994. ACM.

[98] Russell Smith. *Intelligent Motion Control with an Artificial Cerebellum*. PhD thesis, University of Auckland, New Zealand, July 1998.

[99] Russell Smith. Open dynamics engine. `http://opende.sourceforge.net/`, 2002. [This open-source software library is still being actively developed by multiple authors].

[100] Russell Smith. Constraints in rigid body dynamics. *Best of Game Programming Gems*, 2008.

[101] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. *ACM Trans. Graph.*, 26(3), July 2007.

[102] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Soncs, Inc, 2006.

[103] Yichao Sun, Rong Xiong, Qiuguo Zhu, Jun Wu, and Jian Chu. Balance motion generation for a humanoid robot playing table tennis. In *Humanoid Robots (Humanoids), 11th IEEE-RAS International Conference on*, pages 19–25, oct. 2011.

[104] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, August 1999.

[105] Graham W. Taylor, Geoffrey E. Hinton, and Sam Roweis. Modeling human motion using binary latent variables. In B. Schoelkopf, editor, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 1345–1352. MIT Press, 2007.

[106] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH Comput. Graph.*, 21(4):205–214, August 1987.

[107] Emanuel Todorov and Michael Jordan. Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5(11):1226–1225, 2002.

[108] Emanuel Todorov, Weiwei Li, and Xiuchuan Pan. From task parameters to motor synergies: A hierarchical framework for approximately optimal control of redundant manipulators. *J. Robot. Syst.*, 22:691–710, November 2005.

[109] M. Tournier, X. Wu, N. Courty, E. Arnaud, and L. Revret. Motion compression using principal geodesics analysis. *Computer Graphics Forum*, 28(2):355–364, 2009.

[110] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. *ACM Trans. Graph.*, 26(3), July 2007.

[111] E.T. Turner and W. Clouse. *Winning Racquetball: Skills, Drills, and Strategies.* Human Kinetics, 1996.

[112] Marsette Vona. Hierarchical decomposition and kinematic abstraction with virtual articulations. In Jadran Lenarcic and Michael M. Stanisic, editors,

Advances in Robot Kinematics: Motion in Man and Machine, pages 33–43. Springer Netherlands, 2010.

[113] J.M. Wang, D.J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):283–298, feb. 2008.

[114] Mike Vande Weghe, Dave Ferguson, and Siddhartha S. Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *Humanoids*, pages 477–482, 2007.

[115] E.C. Whitman and C.G. Atkeson. Control of a walking biped using a combination of simple policies. In *Humanoid Robots, (Humanoids). 9th IEEE-RAS International Conference on*, volume 9, pages 520–527, dec. 2009.

[116] David A. Winter. *Biomechanics and Motor Control of Human Movement*. John Wiley and Sons, New York, fourth edition, 2009.

[117] Andrew Witkin and Michael Kass. Spacetime constraints. *SIGGRAPH Comput. Graph.*, 22(4):159–168, June 1988.

[118] Pawel Wrotek, Odest Chadwicke Jenkins, and Morgan McGuire. Dynamo: dynamic, data-driven character control with adjustable balance. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, Sandbox '06, pages 61–70, New York, NY, USA, 2006. ACM.

[119] Yuting Ye and C. Karen Liu. Optimal feedback control for character animation using an abstract model. *ACM Trans. Graph.*, 29:74:1–74:9, July 2010.

[120] Changxi Zheng and Doug L. James. Toward high-quality modal contact sound. *ACM Trans. Graph.*, 30(4):38:1–38:12, July 2011.

[121] Victor Zordan, Adriano Macchietto, Jose Medina, Marc Soriano, and Chun-Chih Wu. Interactive dynamic response for games. In *Proceedings of the 2007 ACM SIGGRAPH symposium on Video games*, Sandbox '07, pages 9–14, New York, NY, USA, 2007. ACM.

[122] Victor B. Zordan and Jessica K. Hodgins. Tracking and modifying upper-body human motion data with dynamic simulation. In *Computer Animation and Simulation 99*, pages 13–22. Eurographics, 1999.

[123] Victor Brian Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 89–96, New York, NY, USA, 2002. ACM.

[124] Victor Brian Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic response for motion capture animation. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 697–701, New York, NY, USA, 2005. ACM.

[125] Victor Brian Zordan and Nicholas C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 245–250, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.