Automated Discovery and Learning of Complex Movement Behaviors

Igor Mordatch

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2015

Reading Committee:

Emanuel Todorov, Chair

Zoran Popović

Dieter Fox

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Automated Discovery and Learning of Complex Movement Behaviors

Igor Mordatch

Chair of the Supervisory Committee:
Associate Professor Emanuel Todorov
Computer Science & Engineering

In order to create useful physical robots, tell narratives in animated films and interactive games, or understand the underlying principles behind human movement, it is necessary to synthesize movement behaviors with the same wide variety, richness and complexity observed in humans and other animals. Moreover, these behaviors should be discovered automatically from only a few core principles, and not be a result of extensive manual engineering or merely a mimicking of demonstrations. In this thesis, I develop novel optimal control methods and apply large-scale neural network training to synthesize movement trajectories and interactive controllers that give rise to a range of behaviors such getting up from the ground, climbing, moving heavy objects, hand manipulation, acrobatics, and various cooperative actions involving multiple characters and their manipulation of the environment. Coupled with detailed models of human physiology, motions that match various kinematic and dynamic aspects of real human motion can be produced *de novo*, giving the predictive power to conduct virtual biomechanics experiments. The resulting movements are also used to successfully control a physical bipedal robot. The approach is fully automatic and does not require domain knowledge specific to each behavior, pre-existing examples or motion capture data. Although discovery and learning are computationally-expensive and rely on cloud and GPU computing, the interactive animation can run in real-time on any hardware once the controllers are learned.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my adviser Emo Todorov and co-adviser Zoran Popovic for providing inspiration, guidance and encouragement throughout my graduate studies. I would also like to thank my collaborators on the projects included in this thesis, Jack Wang, Vladlen Koltun, Kendall Lowrey, and Galen Andrew. Without their hard work, much of this thesis would not have been possible. Additionally I am very grateful to my friends inside and outside of the department for my academic and personal growth.

And lastly, I want to thank my family who have given me their unwavering love and support throughout this journey.

Chapter 1

INTRODUCTION

Automated synthesis of complex movement behaviors is one of the long-standing grand challenges in computer graphics and robotics, that would also have a great impact on biomechanics, and neuroscience. Human and other animal motor behavior exhibits a seemingly infinite complexity at a multitude of scales, from the movement of a hand to the movement of multiple interacting animals. Building competent and useful physical robots, or visually telling narratives in virtual media such as animated films and interactive games both require endowing movements of robots or virtual actors with the same level of richness and expressiveness observed in animal motor behavior. Conversely, ability to artificially synthesize such motor behavior from first principles can give scientific insight into how it comes about in biological systems.

An automated synthesis method would ideally be capable of creating motions that span the space of all possible animal behaviors. In addition, such a method would not require expert or labor-intensive authoring in the form of keyframes, reference trajectories, or any specific details of the intended movement. It would also not require any direct or indirect use of motion capture data. Instead, movement details and complexity should emerge *de novo* from an automated procedure whose only inputs are intuitive high-level goals that are easy to specify.

For example, the user would like to synthesize realistic walking for a given human character by simply specifying a desired velocity for the torso. Running should emerge when the specified velocity is higher, jumping should emerge when a target height is given, and kicking when the character's foot must have certain velocity at a given point in space. In the task of a hand picking up an object, user merely specifies the desired final position of the object. The hand and finger movements and finger-object interactions are then synthesized automatically. Similarly, twirling a pen between the fingers is specified as a desired angular velocity for the pen; drawing is specified

as a desired trajectory for the tip of the pen, etc.

Very importantly, an automatic framework for synthesis of complex behaviors should not be restricted to human morphology. After all, many robotic platforms bear no resemblance to humans. The framework should be capable of creating complex behaviors for arbitrary real or invented morphologies, creating a rich gamut of behaviors that spans all imaginable interactions with the environment, and all imaginable movement goals.

The most progress towards this ambitious agenda has been made in the domain of walking. After three decades of intensive research, we now have algorithms that can make simulated humanoids walk robustly and realistically in response to high-level interactive inputs such as desired body velocity and orientation. These algorithms are successful because they exploit domain-specific knowledge: state machines synchronized to the relatively simple and stereotypical pattern of foot-ground contacts, reduced models based on inverted-pendulum dynamics or other features important for walking, and trajectory optimization methods that rely on customized cost functions and manual specification of contacts. The success of these methods comes at the price of limited generality: they provide a unique and different type of model and solution for each of the common locomotion tasks such as walking, running and jumping. With the state-of-the-art in automated motion synthesis prior to this thesis, any additional complex behavior would require a new movement model carefully crafted by experts from scratch. Each of these new movement models would require specific domain knowledge carefully integrated into the motion synthesis algorithm.

Despite its importance, current understanding of hand manipulation lags even further behind locomotion. This likely reflects the intrinsic difficulty of the problem. Indeed it is notable that most animal species are capable of impressive locomotion, while only humans can perform full-blown manipulation, and appear to do so using specialized neural pathways that partially bypass the spinal circuits [100]. From a computational perspective hand manipulation is a difficult problem not only because of the large number of degrees of freedom of the hand and under-actuation of the object being manipulated, but perhaps more importantly, because of the contact interactions involved. Planning and control in the presence of contact discontinuities is very challenging. Nevertheless hand manipulation is unique in that a large number of contacts can appear and disappear in rapid

succession, and dynamic phenomena such as rolling and sliding can be actively used. This makes it virtually impossible to synthesize realistic hand manipulation by pre-specifying contact events as in the case of walking.

An alternative to automatic synthesis is to manually animate motions from scratch, but this requires a skilled artist and is a very time and labor intensive process. Similarly, approaches based on motion capture or scripting have played an important role in computer graphics and will continue to do so, but do not easily generalize to novel behaviors or body morphologies. Motion capture is alsocparticularly technically challenging in hand manipulation: optical systems suffer from occlusions and marker mis-identifications when tracking a large number of markers in a small volume, while data gloves impede natural movements and also limit tactile sensation – which in turn is essential for dexterous manipulation in humans [53].

The behavior-specific approach to motion synthesis and capture is at odds with the variety of human motor behavior, exhibited in movements that do not fall into standard categories such as locomotion or reaching. Often, these behaviors are more challenging than locomotion in the sense that they involve longer, more complex (spatially and temporally) movement plans, and less stereotypical or cyclic movements. Examples include movements that use more than just legs, or just arms, but that use other parts of the body, movements that can represent the full space of interactions between the human body and the ground and other arbitrary objects in the environment, complex manipulations of objects by one or many humans collaboratively, hand-walking, climbing – to name just a few. In this thesis I present a step towards a more general yet fully automated framework for behavior synthesis, capable of producing a wide variety of less commonly studied but important animal behaviors.

1.1 Contact-Invariant Trajectory Optimization

A very general approach to motion synthesis is to pose the problem as *trajectory optimization* [140]. The trajectory of the robot or virtual character is treated as a collection of unknown parameters that must satisfy objectives or constraints such as kinematic and dynamic plausibility of the motion, along with high-level goals specified by the user. Given a particular initial position of the

robot and a goal, traditional numerical optimization methods can be used to solve for an optimal trajectory that simultaneously satisfies all these objectives.

The problem is challenging for all but the simplest problems because trajectory optimization on complex three-dimensional humanoid models lead to high-dimensional search spaces and complex nonlinear constraints that are difficult to satisfy. Furthermore, ground contact forces that change abruptly with body kinematics lead to discontinuous objective energy landscapes prone to poor local minima. To alleviate these issues, many approaches make use of human motion capture data to restrict the search space around stereotypical motions.

The power of the framework I propose in a part of this thesis lies it its ability to optimize movements trajectories through multiple contact events. The approach was motivated by the observation that contact interactions are essential for most animal and human movements. Previous work on motion synthesis through numerical optimization has either pre-defined the contact interactions, or expressed them as functions of the movement trajectory and thereby optimized them indirectly, almost as a side effect of trajectory optimization. The reasoning is that, if contacts are essential, they should play a more central role.

At the core of the framework is the contact-invariant optimization (CIO) method that enables simultaneous optimization of contact and behavior. This is done by augmenting the optimization problem search space with scalar variables that indicate whether a potential contact should be active in a given phase of the movement. These auxiliary variables affect not only the cost function but also the dynamics (by enabling and disabling contact forces), and are optimized together with the movement trajectory. Intuitively, CIO is a way of reshaping a highly discontinuous and local-minima-prone search space of movements and contacts, into a slightly larger but much better-behaved and continuous search space that enables local optimization strategies to find good solutions.

The resulting framework is capable of producing a wide variety of important human behaviors, such as getting up from the ground, crawling, climbing, moving heavy objects, acrobatics (handstands in particular), and various cooperative actions involving two characters and their manipulation of the environment. The approach is not specific to humans, but can synthesize behaviors

for other imagined morphologies. It is also be applied to a wide range of hand manipulation tasks including grasping and picking up objects, twirling them between the fingers, tossing and catching, drawing, obtaining motions whose complexity exceeds prior results obtained automatically.

1.2 Control Policy Search:

While the trajectory optimization framework proposed above can solve very complex motion synthesis problems, the process of numerical optimization takes a long time for high-dimensional characters and the resulting solution is appropriate to only one particular initial condition and goal. If a new goal is given, the optimization problem must be solved again. This prevents the method from being applicable to interactive real-time use. Moreover, the synthesized motions often exhibit high variance, where either a small change to the goal or initial condition leads to very large change in the solution motion trajectory. High variance can also arise where individual elements within a cyclic trajectory (such as footsteps in a walk) all looking slightly different from each other. This can make the approach unpredictable when deployed on physical robots. Lastly, trajectory optimization is unlikely to reflect how animals reason about movement. To walk forward, it is unlikely that animals plan intricate orchestration of muscle activities, but instead make use of simple learned motor skills or heuristics that are reused in the behavior.

Therefore, instead of finding an optimal motion trajectory, one can find an optimal *control policy*: a function mapping states to actions of a character, that when applied repeatedly will achieve the given goals. The *policy search* problem is then to find such a policy function. The resulting controllers specified by this policy function can then be used interactively in real-time to generate complex, stable and realistic movements for the character without a significant computational cost. Such controllers can adapt to unexpected changes in goal objectives, perturbations due to noise and modeling errors, and can also serve as computational models in biomechanics and neuroscience .

Unfortunately, creating such controllers to date has been a difficult and largely manual process. The general approach of posing policy parameter search as an optimization problem similar to the previous section leads to very difficult problems for complex tasks. Policy gradient methods derive the gradient of policy performance with respect to parameters and apply numerical optimization

algorithms [95], but are subject to many numerical and local minima issues. Sampling-based methods have also been applied [122], but for complex problems the state and action spaces are often too large to sample thoroughly. Thus, most successful approaches for tasks relied on interpolation in motion capture datasets, as well as state machines designed for specific goals. In contrast, I seek a representation that does not require significant manual tailoring for application to new characters/goals.

Compare this to the rapid march towards automation taking place in machine learning and related areas. It is now possible to train neural networks to perform vision or speech recognition tasks with remarkable accuracy [31, 61, 108]. Equally remarkable is the level of automation and the simplicity of the training procedures. So why are we not enjoying the same benefits in control? It is not because people have not tried. Indeed control was among the early applications of neural networks [88]. However, unlike in vision or speech recognition the learning task for the network is harder. This is because we need precise real-valued outputs as opposed to categorical outputs, and also because our network must operate not on i.i.d. samples, but in a closed loop, where errors can amplify over time and cause instabilities. Another challenge caused by limited datasets is the potential for over-fitting and poor generalization. Reinforcement Learning and Approximate Dynamic Programming [9,109,118] have given rise to many techniques for function approximation in the context of control. These include indirect methods based on value function approximation [65] as well as direct policy search methods [95, 117]. Some of the most interesting robotic behaviors generated by neural networks are swimming and crawling [50] but they do not involve control hazards such as falling, dropping an object or hitting an obstacle. Data-intensive methods have also been pursued in *imitation learning* (alternatively learning from demonstration) [106]. These methods learn policy parameters from demonstration trajectories. In many cases, this amounts to performing regression or classification, however the learned policy is limited to performing tasks similar to demonstration data. And as demonstration trajectories commonly come from humans performing a task, they may not be consistent dataset to learn from.

As part of this thesis, I propose to combine the benefits of trajectory optimization and recent supervised learning methods. The obvious approach would be to generate a lot of optimal trajec-

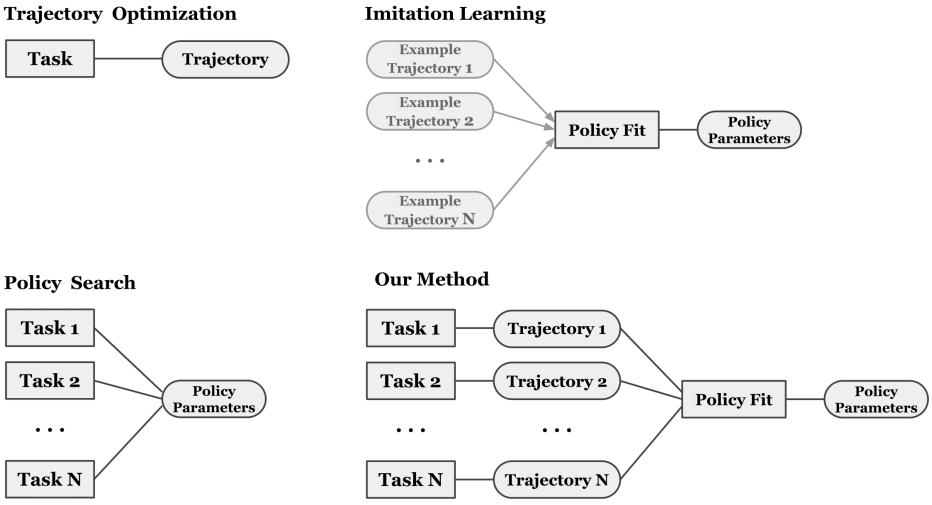


Figure 1.1: Factor graphs of various motion and policy synthesis methods compared to method in this thesis. Parameters being optimized are in circles and objective factors are rectangles.

tories (for different starting states) and use them as training data – similar to imitation learning but with synthetic instead of motion capture data. This is related to DAGGER [103], which I will show later in this thesis does not perform as well as the type of approach I advocate. Straightforward learning from optimized trajectories is problematic because they may be an overly complex, or even inconsistent dataset to learn from. In general, if learning from trajectories was going to produce good controllers for complex tasks, that would have already happened since human motion capture data has been abundant for a while.

At the core of the alternative approach I propose is to merge both trajectory optimization and neural network training together, considering the trajectory optimizer as a teacher rather than a demonstrator. A good teacher adapts to the student, providing guidance in small steps which the student is able to assimilate. Here this will be done by modifying the cost for the trajectory optimizer, so that it still solves the control problem but in a way similar to how the student (fails to) solve it. The method generates a collection of optimal trajectories for a set of tasks (using trajectory optimization), jointly with learning a policy that can represent all these generated trajectories. Such coupling enables guides or regularizes optimal trajectory solutions to become an easier and

consistent dataset to learn a policy from. This approach as related to others is shown in figure 1.1

The resulting algorithm and controller architecture provide interactive real-time control for multiple to complex three-dimensional characters. The same method is able to generate stable and realistic behaviors in a range of diverse tasks – swimming, flying, biped and quadruped walking. It does not require motion capture or task-specific features or state machines. It is also responsive to changes in the task itself. If the task has real-valued parameters – such as the location of a target, or the desired speed of movement – these parameters are encoded as inputs to the network. In this way the same controller can be used to solve a continuum of related tasks.

1.3 Applications to Biomechanics and Robotics

Ability to artificially synthesize complex motor behavior from first principles can give insight into how such behavior comes about in biological systems. It can be used to conduct virtual experiments, for example studying how various surgical procedures, equipment, or prosthetics affect the movement of the human body.

Unfortunately, directly applying trajectory optimization for generation of human-like motion is problematic due to the difficulty of formulating general constraints or objectives that characterize ”human-like” motions. Since the actuation mechanism of the humanoid model has significant impact on the realism of synthesized motion [73, 137], my work actuates the lower body joints with Hill-type musculotendon units (MTUs). The MTUs regularize the synthesized torque patterns and enable the use of a biologically-plausible effort model, such as metabolic energy expenditure [10]. Unlike prior work that utilized MTUs and biologically-inspired objectives to generate human locomotion [137], work in this thesis need not rely on a task-specific control structure. As a result, the approach is not limited to steady-state walking and running. Attention is restricted to the fidelity of lower limb motion since a simplified model of the musculoskeletal system for the lower body is readily available.

The work in this thesis demonstrates that a single trajectory optimization formulation can produce high-fidelity lower body motion for a range of human activities, without the need for prior motion data or task-specific control structures. I evaluate the presented approach by synthesiz-

ing high-fidelity locomotion in a variety of conditions and validate that straight line walking and running results compare favorably to human data in both kinematics, joint moments, and ground reaction force patterns. Additionally, I verify that the results exhibit important features of human motion for walking up and down slopes, gait initiation, as well as jumping and kicking motions.

In the field of robotics, the mechanical and actuation capabilities of modern robots are quickly surpassing our ability to control them in ways that do justice to their hardware. Especially challenging are under-actuated systems such as humanoids where some form of dynamic planning appears to be unavoidable. However, much of previous work on controllers deployed on physical systems again tended to rely on task-specific simplifications and reduced models. Examples are ZMP and inverted pendulum models – which have been very successful in walking [13, 20, 58, 112, 131]. Another pragmatic approach is for a team of engineers to spend weeks designing and fine-tuning a specialized controller (usually a reference trajectory with PD gains around it), but ideally this effort will be automated.

This thesis presents initial work on transferring full-body dynamic movement plans from simulation to the real world. The obvious place to start is to build the most accurate model possible with reasonable effort, optimize a movement trajectory, play it on the robot and see what happens. Not surprisingly, the robot falls. This thesis focuses on working with Darwin-OP, and indeed using an inexpensive and easy-to-handle robot may be key to making progress in this direction. The next obvious step is to use the locally-optimal time-varying linear feedback control law which most trajectory optimizers generate, and apply it to the physical system with the hope that it will suppress deviations from the planned motion. This improves performance somewhat as will be sown later, but in a nutshell it still doesn't work. One answer could be that more accurate models and state estimators are needed – and if one had them performance would certainly improve. In that sense, the next step in model-based control may not be so much about control but rather about system identification and state estimation. Yet developing more accurate modeling techniques, especially for complex robots where multiple poorly-understood factors can mask each other and complicate the interpretation of sensor measurements, will take time.

My main contribution in this area is a new method for transferring model-based controllers

to physical systems that can tolerate the errors in our existing models. The idea is conceptually simple: instead of optimizing through one (nominal) model, I optimize through an ensemble of models obtained by perturbing the parameters of the nominal model. The new method is again based on contact-invariant optimization and adapted to the ensemble case. I show that the resulting trajectories can indeed be executed on the physical system with high success rate, for diverse tasks including forward and sideways walking, turning, and getting up from the floor.

1.4 Contributions and Thesis Overview

The remainder of this thesis will be structured as follows. Chapter 2 will review the related work in the areas of computer graphics, robotics, machine learning, biomechanics and neuroscience. Chapter 3 will introduce the first contribution of this thesis, contact-invariant optimization (CIO) and describe it's application of simplified character motion synthesis and hand manipulation. Chapters 4 and 5 will further describe the applications of CIO to synthesis of biomechanically-plausible human motion and for motion of physical robots. Chapter 6 will describe the second contribution of this thesis, the method to learn interactive neural network controllers based on results of contact-invariant optimization.

Chapter 2

RELATED WORK

The problem of synthesizing motion behaviors has generated a large amount of interest and proposed solutions in a number of fields, including computer graphics, robotics, and biomechanics. In this chapter, I will review the relevant approaches from these fields.

2.1 Motion Planning and Trajectory Optimization

Early approaches to synthesizing human motion in have been able to produce a wide repertoire of skills [44, 45, 141], but required expert manual specification for each new task. Since then, a lot of fruitful work has focused specifically on the task of locomotion. Simplified dynamical systems representing walking [55,63], running [107], push recovery [99] and uneven terrain navigation [80] have been proposed that are well suited to analysis and control. The results were extended to full-detail bipeds [28, 143, 147] and quadrupeds [23]. [89, 110] have tried to capture different modes of locomotion within a single controller, though they still assume a fixed pattern of foot contacts that are specific to locomotion. Others proposed to intelligently combine individual controllers [21, 26, 34, 91], although they still rely on existence of base controller libraries.

A more general approach to motion synthesis has been to pose the problem as trajectory optimization, subject to user constraints [139]. However, it is a very difficult optimization problem, prone to many local minima and discontinuities due to contact phenomena. To alleviate these issues, many approaches make use of human motion capture data to restrict the search space around stereotypical motions and pre-specify contact events [35, 76, 97, 104]. [75] adapts motion capture data and contacts to multi-character interactions. By optimizing contact times for cyclic patterns, [134] is able to synthesize gaits for a wide variety of non-humanoid characters from scratch. While the key advantage of my method is that it does not need motion capture data, it can never-

theless be adapted to take advantage of available data. All the method would have to do is augment or replace the abstract task specification with one that encourages staying close to the data.

Rather than placing restrictions on the optimization problem to solve it, several methods have focused on shaping the energy landscape to be smoother and better behaved. To widen the solution feasibility region, [130] initially use unphysical helper forces to aid the character and reduce them as optimization progresses, while [148] parametrizes the difficulty of the task itself. [12, 125] reformulate contact as a smooth, rather than discontinuous phenomenon, where contact forces are always active, but smoothly diminish with the distance to the ground. The latter method has demonstrated success for continuous optimization of cyclic gaits [33]. [52] accurately models characters with soft tissue, and shows improvements in stability and robustness for simple tracking algorithms. However, applying these formulations to general trajectory optimization problems remains difficult, because contact decisions are made implicitly as a (highly non-linear) function of the character's pose. Some methods (such as [72, 90, 145]) directly include contact forces as variables to be optimized, but they only have limited ability to manipulate contact state. By contrast, contact variables introduced in this thesis make contact decisions explicit in the optimization problem.

Reasoning about contact events for navigation and object manipulation has also been considered by several planning approaches in robotics. [17, 41, 57, 62] decompose the problem into two stages, first planning foot or hand placements and then synthesizing a motion trajectory that follows those placements. The two stages are only loosely coupled, and may not always produce the most efficient strategies. By contrast, my approach jointly plans both the contact events and the motion trajectory, and is able to exploit any synergies between the two.

The importance of temporally-extended actions and their potential to speed up optimization has been recognized in reinforcement learning, and has led to the Options framework [118] which uses the formalism of Semi-Markov Decision Processes. In that framework however the temporally-extended actions have to be specified in advance, while approach in this thesis discovers them automatically.

2.2 Hand Manipulation

In graphics and animation, the most common approach to hand manipulation has been to rely on motion capture data to various degrees. Collecting such data is not easy as already mentioned, but the lack of automated methods has left animators with no other choice. Among the more popular methods in this class are [96, 144]. I will not discuss the details here (except for the issue of dimensionality reduction – see below) since my work is quite different. An up-to-date discussion of hand manipulation based on motion capture can be found in [146].

Another approach that can synthesize finger-object contacts automatically, given only the desired motion of the object and an initial grasping pose, is [71, 72]. This is a hierarchical scheme where contact forces are computed on the high level while corresponding joint torques are computed on the low level. The high level did not anticipate the making and breaking of contacts, but instead planned forces at existing contacts so as to achieve the desired object accelerations. Rich yet unanticipated contact dynamics were then obtained in forward simulation. The same group later developed a more systematic method for generating contact events, based on careful sampling [146]. The object and wrist trajectories were measured using motion capture, and finger trajectories compatible with the measurements were then computed automatically. The results were quite remarkable. However this method is sensitive to the specifics of the motion capture data, e.g. having irregular object velocity consistent with the underlying finger gate. Unlike methods using soft costs (which can smooth out measurement imperfections), the sampling method uses a hard criterion for compatibility with the recorded motion and makes binary decisions based on this criterion, thus it is not clear how its sensitivity can be reduced.

In the field of robotics, grasping has received more attention than any other type of hand movement, and has been automated through a variety of methods that do not rely on motion capture. These methods are not applicable to dexterous object manipulation, but nevertheless are based on important ideas. One such idea is force closure – which is in some sense the equivalent of locomotion stability criteria such as ZMP. More generally, researchers have considered a variety of grasp quality metrics, and optimized the grasp pose with respect to them [86, 87]. See also the review

papers [11, 93].

A more intuitive yet robust approach to grasp synthesis is to position and pre-shape the hand near the object, and then simply close the fingers, relying on their compliance to curl around the object [25]. In this approach one does not need to model the hand-object interactions in detail. The quality of an initial pose can be defined as the grasp quality of the final pose resulting from this procedure. Even if such heuristics can work, it is not clear how often humans actually use them. Indeed it was recently shown that the nervous system predicts finger-object contacts around 100 msec before they occur, and begins to adjust muscle activity so as to transition from motion to force [78]. In terms of numerical optimization, it is notable that methods for automated grasp synthesis have relied on extensive sampling or restarts, again because the search space has many poor local minima.

While biological systems have many DOFs, the movement behaviors observed in daily life span a small fraction of the space that is potentially accessible. This is due to a combination of biomechanical and neural factors which have been debated for decades [64, 128]. Every researcher who has looked for dimensionality reduction in movement data appears to have found it - at the level of kinematics [92, 105, 124], instantaneous muscle activity [129], spatio-temporal patterns of muscle activity [27], organization of feedback control laws [77]. This may be why it is possible to interpolate in motion capture databases and obtain plausible movements. Without reduced dimensionality, the volume of space that needs to be filled with data would be so large that interpolation would be unlikely to work. With regard to hand manipulation, it was shown [105] that the hand poses used to grasp arbitrary objects lie in a surprisingly low-dimensional space: the first 4 principal components accounted for over 95% of the variance in the data. In more complex manipulation tasks the effective dimensionality can increase from 4 to 10, but is still half the number of recorded DOFs [124]. This low-dimensional structure has been exploited in robotics to speed up the search for viable grasping poses [18].

2.3 Biomechanically-Plausible Motion Synthesis

As mentioned in the first section, early efforts in human-like motion synthesis focused on hand-designing controllers for specific activities and significant efforts in recent years being devoted to improving controller robustness and motion style for simulated locomotion. Yet methods that produce the most human-like motions require significant customization of the target trajectory or the control structure [22, 137], while more flexible control algorithms fall short of generating motions that appear natural [28]. For example, [137] demonstrate that human-like walking and running can emerge from biologically-plausible modeling of the musculotendon units in the lower body. However, their work relies on a detailed control structure that was designed specifically for steady-state locomotion and produced only fixed-velocity walking or running.

While trajectory optimization had shown early promise with simple character models [19, 140], applying it to full three-dimensional human models proved challenging. Traditional trajectory optimization formulations have failed to produce high-fidelity human motion de novo, both due to the difficulty of the optimization problem, and the difficulty of formulating general constraints or objectives that characterize "human-like" motions. Success again relied on using motion capture or prior animation data to constrain the problem, or limiting the scope to ballistic motions that are determined by relatively simple mechanical principles [35, 73, 74, 97, 104, 114]. [134] presented impressive results for imaginary animal locomotion, but their work did not produce high-fidelity human gaits. Similarly [3] explored a wide range of humanoid motions, but their work relied on many task-specific objectives and produced results of limited fidelity.

In biomechanics, trajectory optimization with abstract models had been used to study arm and saccadic eye movements [40], as well as human gait selection [111]. Anderson and Pandy [7] used dynamic optimization with a detailed musculoskeletal model of the lower body to synthesize a single walk cycle. However, they dealt with a much more restricted solution space, in which the first configuration in the trajectory was specified based on human walking data and the motion was constrained to be cyclical. The same model was also used to simulate a vertical jump [6], again with a fixed initial configuration. I likewise use a musculoskeletal model for the lower body,

which enables better modeling of energy expenditure and restricts internal forces to be biologically plausible. However, the optimization framework in this thesis is considerably more powerful and supports motion synthesis for a broader range of activities.

2.4 Interactive Controllers

Interactive character control has long been attempted with a multitude of techniques. A common approach is to interpolate existing kinematic animation clips or build statistical models of various levels of sophistication to generate novel motions at interaction time [39, 59, 68, 70, 102, 135]. However, these kinematic techniques do not consider physical dynamics: they are not able to respond to new disturbances and may require additional work to clean up kinematic artifacts (such as foot sliding [60]). Additionally, they require prior data representing motion instead of generating behavior directly from a creature's structure. [133] demonstrate novel creature action with motion graphs, but contact characteristics still need to be specified for each creature. In a slightly different direction, datasets manually animated in creature-invariant coordinates can be used for previously unspecified creature animation [42].

Using physics based controllers allows for interaction, but once again, these controllers need specially designed architectures for each range of tasks or characters. Smooth and low-dimensional dynamical systems such as quadrotors admit principled solutions based on nonlinear control theory [47, 83]. Biped location common approaches include specialized state machines and simplified models [**?**, **?**, **?**, **?**, **?**, 99, 149]. For quadrupedal characters, a different set of state machines, contact schedules and simplified models is used [24, 101]. For flying and swimming yet another set of control architectures, commonly making use of explicit cyclic encodings, have been used [51, 54, 142]. Work in this thesis seeks to unify these different controller architectures.

In the more general setting, concurrent to the work in this thesis, controller architectures based on neural networks was shown to produce very promising results [49, 69, 119]. Additionally, in some cases trajectory optimization is applicable in interactive settings using model-predictive control [1, 120], where the trajectory is re-optimized as often as possible, using the previous solution for warm-start. This has given rise to impressive results, but is limited to behaviors where short-

horizon planning (on the order of a second) is sufficient, and furthermore if the optimizer gets into a poor local minimum it is not clear how to proceed in real-time.

Chapter 3

DISCOVERY OF COMPLEX BEHAVIORS THROUGH CONTACT-INVARIANT OPTIMIZATION

This chapter will present a motion behavior synthesis framework capable of producing a wide variety of important human behaviors that have rarely been studied, including getting up from the ground, crawling, climbing, moving heavy objects, acrobatics (hand-stands in particular), and various cooperative actions involving two characters and their manipulation of the environment. The framework is not specific to humans, but applies to characters of arbitrary morphology and limb configuration. The approach is fully automatic and does not require domain knowledge specific to each behavior. It also does not require pre-existing examples or motion capture data.

At the core of the framework is the contact-invariant optimization (CIO) method introduced in this chapter. It enables simultaneous optimization of contact and behavior. This is done by augmenting the search space with scalar variables that indicate whether a potential contact should be active in a given phase of the movement. These auxiliary variables affect not only the cost function but also the dynamics (by enabling and disabling contact forces), and are optimized together with the movement trajectory. Additional innovations include a continuation scheme allowing helper

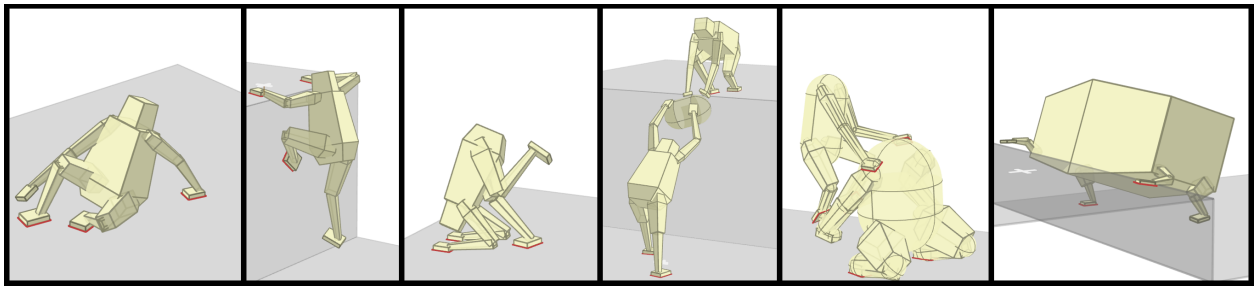


Figure 3.1: A selection of motions synthesized by Contact-Invariant Optimization.

forces at the potential contacts rather than the torso, as well as a feature-based model of physics which is particularly well-suited to the CIO framework.

3.1 The key idea: Contact-Invariant Optimization (CIO)

As with prior methods for automated behavior synthesis, the CIO method also comes down to exploiting domain-specific knowledge. The important difference is that the domain to which the method is tailored is much larger, and includes any behavior of any articulated character where contact dynamics are essential. This is a very large domain because almost all limb movements performed on land are made for the purpose of establishing contact with some object (including the ground) and exerting forces on it. A suitable set of contacts provides actuation: once you grasp an object you can manipulate it; once you plant your feet on the ground you can generate forces on your torso. Complex movements tend to have phases within which the set of active contacts remains invariant. Such invariance greatly reduces the space of candidate movements. In complex behaviors and in complex environments, however, it is difficult to know in advance what these contact sets should be and how they should change from one phase to the next. Unlike prior work on walking where contact information was specified manually and left outside the scope of numerical optimization, I believe that discovering suitable contact sets should be the central goal of optimization. Once this is done, optimizing the remaining aspects of the movement tends to be relatively straightforward.

Intuitively, CIO is a way of reshaping a highly discontinuous and local-minima-prone search space of movements and contacts, into a slightly larger but much better-behaved and continuous search space that enables optimization strategies to find good solutions. The main technical innovation in the CIO method is the introduction of scalar variables that indicate whether a potential contact should be active in a given phase. These auxiliary variables affect not only the cost function but also the dynamics (by enabling and disabling contact forces), and are optimized together with the movement trajectory. In this way the optimizer is provided with abstract but nevertheless useful information: namely that movements should have phases, and that the contact set should remain invariant within each phase. The specific sets of active contacts that are suitable for each phase

of each behavior are then discovered by the optimizer fully automatically. Additional innovations include a continuation scheme allowing helper forces at the potential contacts rather than the torso, as well as a feature-based model of physics which is particularly well-suited to the CIO framework.

3.2 Rationale and Overview

My work was motivated by the observation that contact interactions are essential for most animal and human movements. Previous work on motion synthesis through numerical optimization has either pre-defined the contact interactions, or expressed them as functions of the movement trajectory and thereby optimized them indirectly, almost as a side effect of trajectory optimization. My reasoning was that, if contacts are essential, they should play a more central role. This suggested optimizing over some auxiliary decision variables which directly describe when and where contacts are made. Our first attempts to develop such a method failed in interesting ways. We defined discrete variables specifying contacts between pairs of objects, and a continuous feedback control method that pushed the character along trajectories consistent with this high-level contact specification. I found that, even though setting such discrete variables was relatively easy for humans (resulting in a novel way of motion scripting), optimizing them automatically was basically intractable. This was partly because discrete optimization is generally hard, and partly because it is difficult to tell what constitutes a good set of contacts without simultaneously considering the detailed trajectory that instantiates them. The moral of the story was that decisions regarding contact interactions should be encoded as continuous rather than discrete variables, and should be optimized simultaneously with the movement trajectory.

Continuous specification of desired contacts naturally leads to the idea of weights in cost functions. The contact-related auxiliary variables I use here (denoted $c_i \geq 0$ for contact i) have the following semantics: if c_i is large contact i must be active (i.e. the corresponding bodies must be touching), while if c_i is small I do not care what happens at contact i. Another important observation is that complex behaviors are naturally decomposed into phases, and the set of contacts remains invariant in each phase. The contact forces are not invariant (on the contrary, they change a lot) but the presence or absence of a contact is. This suggests making $c_{i,t}$ piece-wise constant over

time, i.e. defining $c_{i,\phi(t)}$ where $\phi(t)$ is the movement phase at time t. The most direct approach at this point would be to define auxiliary cost terms weighted by $c_{i,\phi}$. If I did that, however, the optimizer will immediately set all c's to zero and effectively eliminate my auxiliary costs. One way to prevent this would be to constrain the sum of the c's, but this amounts to telling the optimizer how much overall contact it should use in a given behavior, and I do not know the answer in advance.

This leads to the next important realization – which is that my auxiliary variables c should also affect the dynamics, in such a way that setting them to zero would be suboptimal. Since they are associated with contacts, the natural way to enter the dynamics is to allow contact forces to be generated at contact i only when the corresponding c_i is large. This has another unexpected benefit: instead of finding the active contacts and performing various calculations that depend on the output of the collision detector (and are therefore non-smooth and difficult to optimize over), we can assume that the active contacts are those whose c's are large, resulting in simpler computations with smooth output.

The approach outlined above requires all potential contacts to be enumerated in advance, and an auxiliary variable c_i to be defined for each of them. If there are N bodies, we have N^2 potential contact pairs. To keep the problem size more manageable I introduce an unusual notion of contact which is one-sided, i.e. we associate the c's with individual bodies (called end-effectors below) rather than pairs of bodies. Thus we have N auxiliary variables instead of N^2. A potential drawback is that the law that "each action has equal and opposite reaction" no longer holds in a strict sense, although in practice the violations appear to be small and we have not found them to be problematic.

Finally, I introduce a simplified physics model consistent with my contact-centric philosophy. Instead of parametrizing the joint-space configuration of the character and using forward kinematics to compute end-effector positions and orientations, we parameterize the end-effectors and use inverse kinematics to define the joint-space configuration. In this way the optimizer can work directly with the end-effectors to which the auxiliary variables are associated. Kinematic constraints (i.e. fixed limb sizes and joint limits) are enforced as costs, and the dynamics are simplified by assuming that all mass is concentrated at the torso. We do not yet know if this physics simplifi-

cation was necessary, and will find out in future work. We do know however that the method as presented below exceeded my most optimistic expectations, and synthesized remarkably complex movements within a couple of minutes of CPU time.

3.3 Contact-Invariant Optimization

I now describe the CIO method in detail. I will begin with a general formulation that can be adapted to different types of physics models and tasks. I then describe my specific simplification of physics, followed by details of the behavioral tasks and the numerical optimization procedure.

3.3.1 General formulation and contact-invariant cost

Let $\mathbf{s}$ denote the real-valued solution vector that encodes the movement trajectory and auxiliary variables. The trajectory can be represented directly by listing the sequence of poses, or by function approximators such as splines (which is what I use here). All that is required is that the character pose $\mathbf{q}_t(\mathbf{s})$ is a well-defined function of $\mathbf{s}$ at each (discrete) point in time $1 \leq t \leq T$. The auxiliary variables $c_{i,\phi(t)}(\mathbf{s}) \geq 0$ are also included in $\mathbf{s}$. The overall movement time T is partitioned into K intervals/phases, and $1 \leq \phi(t) \leq K$ is the index of the phase to which time step t belongs. In my current implementation the number of phases is predefined and their durations are equal, although in principle these parameters can also be optimized in an outer loop. $1 \leq i \leq N$ is an index over "end-effectors". Here end-effector does not refer to an entire rigid body (e.g. a hand or a foot), but to a specific surface patch on one of the rigid bodies. These patches/end-effectors are the only places were contact forces can be exerted, as explained below. The function $\mathbf{p}_i(\mathbf{q}) \in \mathbb{R}^3$ returns the center of patch i.

The CIO method computes the optimal solution $\mathbf{s}^*$ by minimizing a composite objective function $L(\mathbf{s})$ in the form

$$L(\mathbf{s}) = L_{\text{CI}}(\mathbf{s}) + L_{\text{Physics}}(\mathbf{s}) + L_{\text{Task}}(\mathbf{s}) + L_{\text{Hint}}(\mathbf{s}) \tag{3.1}$$

L_{CI} is a novel contact-invariant cost introduced here. L_{Physics} penalizes physics violations; physical consistency is enforced using a soft cost rather than a hard constraint because this enables powerful

continuation methods. L_{Task} specifies the task objectives, and is the only term that needs to be modified in order to synthesize a novel behavior. L_{Hint} is optional and can be used to provide hints (e.g. ZMP-like costs are used here) in the early phases of optimization. Continuation methods are implemented by weighting these costs differently in different phases of optimization; see below.

The contact-invariant cost L_{CI} is defined as

$$L_{\text{CI}}(\mathbf{s}) = \sum_t c_{i,\phi(t)}(\mathbf{s}) \left(\|\mathbf{e}_{i,t}(\mathbf{s})\|^2 + \|\dot{\mathbf{e}}_{i,t}(\mathbf{s})\|^2 \right) \tag{3.2}$$

$\mathbf{e}_{i,t}$ is a 4D contact-violation vector for end-effector i at time t. Recall that a large value of $c_{i,\phi(t)}$ means that end-effector i should be in contact with the environment during the entire movement phase $\phi(t)$ to which time t belongs. Thus when c is large we want the corresponding $\mathbf{e}$ to be small. This vector encodes misalignment in both position and orientation. The first 3 components of $\mathbf{e}$ are the difference vector between the end-effector position $\mathbf{p}_i(\mathbf{q}_t)$ and the "nearest point" on any surface in the environment (including other body segments). The last component of $\mathbf{e}$ is the angle between the surface normal at the nearest point and the surface normal at the end-effector. The cost L_{CI} penalizes both $\mathbf{e}$ and its velocity $\dot{\mathbf{e}}$ which corresponds to slip.

Our definition of "nearest point" is unusual in an important way: we effectively use a soft-min instead of a min operator. Let $\mathbf{n}_j(\mathbf{p})$ denote the actual nearest point to $\mathbf{p}$ on surface j. Define the weights

$$\eta_j(\mathbf{p}) = \frac{1}{1 + \|\mathbf{p} - \mathbf{n}_j(\mathbf{p})\|^2 k} \tag{3.3}$$

where $k = 10^4$ is a smoothness parameter. A virtual nearest point is then obtained by normalizing the weights η_j to sum to 1, and computing the weighted average of the $\mathbf{n}_j$'s. Intuitively, when $\mathbf{p}_i$ is far from any surface (and c_i is large), the cost L_{CI} will push it towards some average of the surface "mass". When $\mathbf{p}_i$ gets close to a surface, it will be pushed towards the nearest point on that surface. This construction also makes L_{CI} smooth, which facilitates numerical optimization.

3.3.2 Inverse dynamics and physics-violation cost

Next I describe the physics-violation cost L_{Physics}. This cost has two components. One is general and depends on my novel formulation of contact dynamics which is essential to the CIO method. The other is specific to the simplified model of multi-body dynamics described below – which can be replaced with a full physics model without modifying the rest of the method. In this subsection I focus on a single time step and omit the time index t.

Let $\mathbf{f} \in \mathbb{R}^{6N}$ denote the vector of contact forces acting on all N end-effectors. We have a 6D vector per contact because I allow torsion around the surface normal, and furthermore the origin of the contact force is allowed to move inside the end-effector surface patch. Thus, unlike previous work which models contact as a sum a multiple contact points, I model an entire contact surface patch, allowing us to reason about contact at a coarser scale and speed up the optimization. Note that all potential contacts are considered at all times, and so the dimensionality of the contact force vector remains constant, resulting in smoothness of the cost. Contact forces here are associated with individual end-effectors and not with pairs of contacting bodies. We only model contacts at pre-defined end-effectors, although my definition is sufficiently general to include surface patches on any body part.

Let $J(\mathbf{q}) \in \mathbb{R}^{6N \times D}$ denote the Jacobian matrix mapping generalized velocities $\dot{\mathbf{q}}$ to contact-space velocities for the contact model described above; $D = \dim(\dot{\mathbf{q}})$ is the number of degrees of freedom in the character. Let $\tau(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ denote the inverse of the smooth dynamics, which equals the sum of contact and applied forces:

$$\tau(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = J(\mathbf{q})^T \mathbf{f} + B\mathbf{u} \tag{3.4}$$

Here $\mathbf{u}$ is the vector of applied forces/controls in the actuated space. They are mapped to the full space by the matrix B, which has zeros in the rows corresponding to "root" forces/torques acting on the torso or on passive objects. The smooth inverse dynamics are

$$\tau(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{3.5}$$

where M is the inertia matrix, C the matrix of Coriolis and centrifugal terms, and $\mathbf{g}$ is gravity.

Our goal now is to complete the inverse dynamics computation, and in particular recover $\mathbf{f}, \mathbf{u}$ given τ, J, B. I do this by minimizing the squared residual in (3.4) subject to friction-cone constraints on $\mathbf{f}$ and quadratic regularization for $\mathbf{f}$ and $\mathbf{u}$. Thus I have the following quadratic programming (QP) problem:

$$\begin{aligned} \mathbf{f}, \mathbf{u} = \quad & \arg\min_{\widetilde{\mathbf{f}}, \widetilde{\mathbf{u}}} \left\| J^T \widetilde{\mathbf{f}} + B\widetilde{\mathbf{u}} - \tau \right\|^2 + \widetilde{\mathbf{f}}^T W \widetilde{\mathbf{f}} + \widetilde{\mathbf{u}}^T R \widetilde{\mathbf{u}} \\ & \text{subject to} \quad A\widetilde{\mathbf{f}} \leq \mathbf{b} \end{aligned} \tag{3.6}$$

The pose-dependent matrix $A(\mathbf{q})$ and vector $\mathbf{b}(\mathbf{q})$ encode the standard pyramid approximation to the friction cone which makes the inequality constraints linear. They also include linear inequality constraints that restrict the origin point of each contact force to the surface patch of its corresponding end-effector. Currently these patches are rectangles. For end-effectors that are allowed to grab (i.e. hands) I omit the friction-cone constraints, allowing both positive and negative normal forces.

The control regularization matrix R is constant and symmetric positive definite. The contact force regularization matrix W depends on the auxiliary variables: W is diagonal and its 6 diagonal elements corresponding to the i-th contact force vector are

$$W_{j,j} = \frac{k_0}{c_i^2 + k_1} \tag{3.7}$$

for all j between $6i-5$ and $6i$. I used $k_0 = 10^{-2}, k_1 = 10^{-3}$. The components of W corresponding to hand end-effectors were four times larger, because hands are generally weaker than feet.

The quadratic program (3.6) is convex and therefore has a unique solution – which can be found using a number of algorithms. Here I used a specialized sequential solver due to [**?**] because it is easy to implement efficiently, although the results obtained with an off-the-shelf QP solver were very similar.

In summary, inverse dynamics are computed by first computing the quantities $\tau(\mathbf{s}), J(\mathbf{s}), A(\mathbf{s}), \mathbf{b}(\mathbf{s}), W(\mathbf{s})$ as described above, and then solving (3.6) which yields $\mathbf{f}(\mathbf{s})$ and $\mathbf{u}(\mathbf{s})$. Re-introducing the time index t, the physics-violation cost is

$$L_{\text{Physics}}(\mathbf{s}) = \sum_t \left\| J_t(\mathbf{s})^T \mathbf{f}_t(\mathbf{s}) + B\mathbf{u}_t(\mathbf{s}) - \tau_t(\mathbf{s}) \right\|^2 \tag{3.8}$$

Note that even if this cost can be reduced to zero by a certain choice of $\mathbf{f}$ and $\mathbf{u}$, the actual $\mathbf{f}$ and $\mathbf{u}$ computed in (3.6) will generally be different because of the extra regularization terms.

Trade-off between L_{CI} and L_{Physics}

Let us now examine how the two cost terms L_{CI} and L_{Physics} defined in (3.2) and (3.8) are affected by the auxiliary variables c. The term L_{CI} achieves its global minimum of zero when all c's are set to zero. However this makes the W matrix in (3.7) large, contact forces become expensive, and the QP solver for (3.6) prefers to violate physics rather than use large contact forces – which in turn leads to a substantial L_{Physics} term. Conversely if all c's are large, the QP solver is able to reconcile any trajectory with the inverse dynamics model by generating contact forces at all end-effectors, including those that are not actually in contact. This makes L_{Physics} small, but now L_{CI} is large because end-effectors that are not in contact have large c's. Thus the optimal trade-off is achieved when c_i is large only for end-effectors that are in contact, and furthermore the trajectory can be generated using contact forces only at those end-effectors. In other words, at the optimal solution the auxiliary variables c correspond to the contacts that end up being active.

Since the controls $\mathbf{u}$ are constrained to act in the actuated space, root helper forces (often used in prior work for continuation) are not allowed here. If we were to allow such forces the above trade-off in terms of c would no longer hold. On the other hand, the ability of my method to generate contact forces from a distance provides an even more effective continuation method: wishful thinking in the early phases of optimization is still possible, but it is always associated with potential contacts, making it easier to transition to physically-realistic contact dynamics later in optimization.

Simplified physics model

While the CIO method as described above can be used with standard physics models as implemented in existing physics engines, my implementation relies on a simplified model yielding a favorable trade-off between physical realism and optimization efficiency. Instead of representing

the pose $\mathbf{q}$ directly and then computing the end-effector positions $\mathbf{p}_i(\mathbf{q})$ using forward kinematics, I represent the end-effector positions as well as their orientations directly (i.e. as functions of spline parameters contained in $\mathbf{s}$) and then define the pose $\mathbf{q}$ using inverse kinematics; see Appendix at the end of this chapter. All mass is assumed to be concentrated at the root bodies: the torso of each character, as well as any passive objects. Non-smooth movements of the (now-massles) limbs are avoided by including an acceleration cost described later. The inverse dynamics are still in the form (3.4) and the quadratic program defining the contact force and control is still in the form (3.6), but all computations are now simplified.

Representing the pose in terms of end-effector positions and orientations makes it difficult to enforce kinematic constraints exactly. However this can be turned to an advantage, by introducing an additional continuation method that allows limbs to stretch and joint limits to be violated early in optimization. This is done by adding quadratic costs to L_{Physics}, that penalize any deviations of the limb lengths from their reference values as well as any joint limit violations. I also penalize penetration of the character's bodyparts (approximated for collision with capsules shown in figure 4.2) against the environment, or other bodyparts.

3.3.3 High-level goals and task cost

The cost $L_{\text{Task}}(\mathbf{s})$ encodes the high-level goals of the movement. It includes task-specific terms specifying the desired outcome, and generic terms (integrated over time) specifying that the movement should be energy-efficient and smooth:

$$L_{\text{Task}}(\mathbf{s}) = \sum_b \ell_b(\mathbf{q}_T(\mathbf{s})) + \sum_t \|\mathbf{f}_t(\mathbf{s})\|^2 + \|\mathbf{u}_t(\mathbf{s})\|^2 + \|\ddot{\mathbf{q}}_t(\mathbf{s})\|^2 \tag{3.9}$$

Here ℓ_b are task-specific terms which only depend on the final pose $\mathbf{q}_T$, and b is an index over different tasks. Several tasks can be composed together, such as combining a standing task with the moving to target task. I use the above general form of L_{Task} for all tasks except for kicking/punching. In that case I specify an ℓ_b at regular intervals when each target should be hit, and also include dependence on $\dot{\mathbf{q}}$ because I want the targets to be hit with a certain end-effector velocity.

The general procedure for constructing the task-specific costs ℓ_b is to identify a vector of positional (and optionally velocity) features $\mathbf{h}_b(\mathbf{q})$ that are key to task b, define the desired feature values $\mathbf{h}_b^*$ at the end of the movement (or at other important points in time such as target hits), and then construct ℓ_b as

$$\ell_b(\mathbf{q}_T(\mathbf{s})) = \|\mathbf{h}_b(\mathbf{q}_T(\mathbf{s})) - \mathbf{h}_b^*\|^2 \tag{3.10}$$

In this way, a final position task ℓ_{pos} can be specified by using $\mathbf{h}_{\text{pos}}$ that selects torso position, and setting $\mathbf{h}_{\text{pos}}^*$ to the desired position. Final orientation task ℓ_{dir} can be defined similarly for torso facing direction. Standing task ℓ_{stand} can be expressed by using a combination of $\mathbf{h}_{\text{stand}}$ and $\mathbf{h}_{\text{stand}}^*$ which specifies that the center of torso should be between two feet, the feet be fully extended, and the torso direction be aligned with the vertical direction vector.

The relative importance of the different features can be adjusted by scaling the corresponding elements of $\mathbf{h}$.

3.3.4 Heuristic sub-goals and hint cost

In the absence of good initialization – which in the present context would correspond to motion capture data or other detailed user inputs I aim to avoid – numerical optimization can be sped up by providing heuristic sub-goals early on, and then disabling them near convergence. Such heuristics (also known as shaping) are not meant to be part of the true cost, but rather guide the solution to a region from where the true cost can be optimized efficiently. I found that even though most of the behaviors I studied could be synthesized without such heuristics, in some cases (particularly those involving two characters) a certain type of heuristic helps. This heuristic is based on the ZMP stability criterion used in locomotion, where the objective is to keep the "zero moment point" $\mathbf{z}(\mathbf{q}, \ddot{\mathbf{q}})$ in the convex hull of the support region [**?**]. Let $\mathbf{n}(\mathbf{z})$ denote the nearest distance (in a soft-min sense) to $\mathbf{z}$ point in the convex hull. I compute $\mathbf{n}$ by expressing it as a convex combination of the end-effector positions: $\mathbf{n} = \sum_i \lambda_i \mathbf{p}_i$ where $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$, and solving for the coefficients λ using quadratic programming regularized by the same weights W as in (3.7). Then

the hint cost is

$$L_{\text{Hint}}(\mathbf{s}) = \sum_t \max\left(\|\mathbf{z}_t(\mathbf{s}) - \mathbf{n}(\mathbf{z}_t(\mathbf{s}))\| - \epsilon, 0\right)^2 \tag{3.11}$$

This is a half-quadratic starting ϵ away from the convex hull. The parameter ϵ is used to adjust how strictly I want to enforce the ZMP stability criterion.

3.3.5 Numerical optimization and continuation

I optimize the composite cost $L(\mathbf{s})$ defined in (3.1) using an off-the-shelf implementation of the LBFGS algorithm. The dimensionality of the vector $\mathbf{s}$ is $(12(N+1)+N)K$, where again N is the number of end-effectors and K is the number of movement phases. The $\mathbf{s}$ particular to my simplified model, is defined in (3.12) and (3.13) of Appendix A. I use K between 10 and 20 depending on the complexity of the task. Each phase lasts 0.5 sec. The inverse dynamics and cost are evaluated at 0.1 sec intervals (note that the analytical spline representation allows us to evaluate the dynamics and cost at any point in time). The gradient $\nabla L(\mathbf{s})$ which is needed for numerical optimization is approximated using finite differences (with $\epsilon = 10^{-3}$). The implementation of finite differences takes advantage of the fact that many of the cost terms depend only on the pose at a single point in time, and do not need to be recomputed when the rest of the trajectory is perturbed.

Continuation is implemented by weighting the four terms in (3.1) differently in different phases of the optimization process (not to be confused with movement phases). The optimization process has three phases as follows. In Phase 1 only L_{Task} is enabled. This causes the optimizer to rapidly discover a movement that achieves the task goals without being physically realistic. In Phase 2 I enable all four terms, except L_{Physics} is down-weighted by 0.1 so that physical consistency is enforced gradually. In Phase 3 I fully enable all terms except for L_{Hint} – which is no longer needed and is undesirable at this point, because I do not want it to affect the final solution. Qualitatively, Phase 1 corresponds to rapid discovery combined with wishful thinking; Phase 2 corresponds to cautious enforcement of physical realism while being guided by optional hints; Phase 3 corresponds to refinement of the final solution. The solution obtained at the end of each phase is perturbed with small zero-mean Gaussian noise (to break any symmetries) and used to initialize the next phase.

The initialization for Phase 1 is completely uninformative – a static initial pose. I found that using such continuation is often important. The good news is that exactly the same continuation scheme was successful in all of the diverse behaviors I studied, and so my method does not need behavior-specific adjustments.

Each stage of the optimization takes between 250-1000 iterations, depending on the complexity of the problem. The total time to synthesize each animation clip ranges between 2 to 10 minutes on a quad-core 2.3GHz Intel Xeon machine.

3.4 Results

By using the tasks ℓ_b described in section 4.3 I can synthesize a wide variety of behaviors.

Getting Up, Walking, Climbing By using only ℓ_{stand} described in section 4.3 and using an initial pose of the character lying on the ground, I synthesize the motion of getting up. Different initial poses (such as lying on the back, or on the stomach) result in different getting up strategies, as seen in the accompanying video. If the character is initially standing, ℓ_{stand} is satisfied by simply continuing to stand. By using ℓ_{stand} task in combination with ℓ_{pos} task and an initially-standing pose, I induce walking. The pattern of foot contacts typical of walking is not specified and emerges automatically from my optimization. ℓ_{pos} task is shown in the corresponding video with a white crosshair.

By using only ℓ_{stand} and ℓ_{pos} tasks, but changing the environment to include an obstacle, a range of strategies emerges from simply stepping over the obstacle, to using hands to prop the character up, to using hands to grip and climb a really tall obstacle. The coefficient of friction μ on the foot contacts is 2 to allow foot plants on completely vertical slopes. Hand contact forces do not have friction cone or positivity constraints to reflect the ability of hands to grip.

Handstands, Punches, Kicks Modifying ℓ_{stand} by swapping feet for hands in the specification and commanding that feet should point upwards, I create a handstand task $\ell_{\text{handstand}}$. When only this task is active and the character is initially standing, they will make a series of preparatory

movements and prop themselves up onto their hands. $\ell_{\text{handstand}}$ can similarly be combined with ℓ_{pos} and ℓ_{dir} tasks.

To generate striking motions such as punches and kicks, I create ℓ_{strike} task that specifies positions and velocities for limb end effectors at various points in time. For punches, I specify random target at every second movement phase (every third phase for kicks). I also add ℓ_{dir} task to make the character face every target. The result is a character striking targets as if they were known in advance, while staying balanced enough to make subsequent strikes.

Non-Human Character Morphologies I also tested my algorithm with characters of different morphologies, such as a biped with a wide torso and quadruped with short legs. The optimization was successful in getting up, walking and climbing scenarios, with strategies appropriate for each morphology. For example, animal trot pattern of contacts (moving front leg and opposite hind leg together) emerges for quadruped walking without explicitly being specified.

Interaction with Objects Additional prop objects can be introduced into the world and their trajectories included as variables in the optimization. An object has auxiliary contact variables for every character's hand, indicating that the end effector is gripping the object. The terms $Ł_{\text{Physics}}$ and L_{CI} now apply to the object as well. The object is able to generate contact forces that move it around, but L_{CI} term for object now requires that hand end effector has to be touching the surface of the object.

Tasks similar to ℓ_{pos} and ℓ_{dir} are used to specify final position and orientation of the object. From only these two tasks, the strategy of the character having to pick up and carry the object to the destination emerges (in particular, no tasks are specified for the character).

Intuitively, the cost terms form the following dependency: to move the object, L_{Physics} requires contact forces, which requires active contacts. Then L_{CI} requires character's hand end effectors to be touching the object. Then limb length constraints require that character be close enough to the object to touch it. This may require walking up to it, and so on.

Interaction Between Characters The algorithm can be extended to jointly optimizing for the motion of multiple characters. For the task of moving the object above, multiple characters distribute the workload and cooperate to pass the object from one to the other. I can control the workload distribution by increasing the penalty on contact forces in equation (3.9) for one of the characters, and making the other character do most of the object carrying work.

Two characters also cooperate to achieve tasks impossible for one, such as ℓ_{pos} for one of the characters specifying a target location above character's height. Because contacts can be made with the surfaces of other characters, the task is achieved by one character climbing on top of the other.

3.5 Discussion and Future Work

In this chapter I presented a fully automated framework for synthesizing a wide range of movement behaviors, and demonstrated its effectiveness on complex and rarely studied behaviors. The framework is agnostic to the morphology of the character, and indeed I showed that movement behaviors can be created for significantly different types of characters. The contact-invariant optimization method could likely be applied to other domains where constraint-driven phases bifurcate the optimization space making it very hard to find solutions numerically. I developed an effective continuation scheme suitable for my optimization method.

I also introduced a feature-based physics model that allows for efficient consideration of dynamic aspects in the inner loop of the optimization procedure. This relaxation naturally comes at a cost. I model the character's kinematics and contact interactions with the environment in detail, but represent its dynamics as a single rigid body and do not model the limb dynamics. This simplification effectively results in limbs with infinitely strong muscles (and no penalty for using them), which in turn can lead to energy-inefficient motions such as using bent knees for support, having arms extended against gravity, or occasional out of place sitting. The latter occurs because sitting minimizes the sum of squared contact forces, but does not consider the effort required to get back up. These limitations may be removed by using full-body inverse dynamics to calculate the character's joint torques, and penalizing the torques or some related quantity.

The key advantage of my framework is the simultaneous optimization of contacts and smooth portions of the movement. This was made possible by introducing an auxiliary decision variable for each potential contact, and keeping these variables constant within each phase. As a result, it was possible to optimize very long and temporally complex movement sequences that have previously remained beyond the reach of numerical optimization methods. The price that had to be paid was that the potential contact points (or rather patches) had to be pre-defined. I am of course penalizing penetrations everywhere, but this is not the same as actively optimizing over active contacts. One way to remove this limitation is to simply increase the number of potential contacts and cover the entire body with sufficient density. It remains to be seen how this will affect CPU time. Another simplification I make is to penalize any relative velocity at contacting end effectors (see (3.2)), which results in trajectories that do not have any noticeable slipping. Instead, in a low friction environment character moves overly conservatively, making sure contact forces do not travel outside the friction cone and is unable to exploit possible slipping when planning motions.

The style of the motions was not the focus of this work, and could use a lot of improvement, particularly for low-energy motions such as walking where humans use every bit of physiology (which I do not model) to their advantage. Since my method performs long-horizon trajectory optimization, I can incorporate biomechanically-inspired cost terms from [137] to shape the stylistic aspects of the motion, which will be the subject of chapter 5.

In all my examples, standard methods for local gradient-based optimization were able to find good solutions efficiently. This is one of the key advantages of my framework. Still, the use of global optimization could provide more robust exploration of the space of motions, especially for tasks such as getting up that have a wide range of possible and equally good solutions. In such cases it would be preferred to produce multiple solutions, and select the ideal one.

In the examples presented here the number and duration of phases was fixed. Generally, I have found no problems in overestimating the number of movement phases required to complete an action. The character typically uses up extraneous movement phases by keeping still or sitting down before or after completing the task. Underestimating the number of phases is more problematic

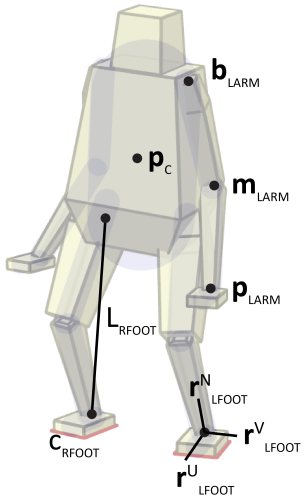


Figure 3.2: Simplified character model used in this chapter. Shown are the features used in my character description with collision capsule geometry overlaid.

and can result in very energy inefficient or completely unphysical leaps in the motion. However, a fixed number of phases would not be as much of an issue if the task costs were reformulated as running costs and the system was used in model-predictive, or online replanning setting. In that case, number of phases would correspond to a future planning horizon, and not dictate the total duration of the motion. The number and duration of phases could also be optimized, although I have not tested this.

Appendix: Simplified Character Model

Our simple model specifies character's state $\mathbf{q}(\mathrm{s})$ at a particular time through a small number of features, rather than a full set of joint angles.

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}_c & \mathbf{r}_c & \mathbf{p}_{1\ldots N} & \mathbf{r}_{1\ldots N} \end{bmatrix}^T \tag{3.12}$$

Where $\mathbf{p}_c$ and $\mathbf{r}_c$ are torso position and orientation, respectively, and $\mathbf{p}_i$, $\mathbf{r}_i$ are end effector positions and orientations for each limb i (see figure 4.2). Rotations are represented with exponential map [38] because of its suitability in trajectory optimization.

From the above features, I can reconstruct the actual character's pose, including limb base locations $\mathbf{b}_i$, which can be derived from local location points on the torso. I assume character's limbs have two links, which allows us to analytically solve for middle joint location $_i$ and orientations of the two links. For limbs that have more than two links, it would be necessary to use an iterative inverse kinematics method to derive the individual joint locations.

I define the motion with positions and velocities of my features at the boundaries between phases. Cubic splines with knots at phase boundaries are used to define a continuous feature trajectory from which positions, velocities, accelerations at any point in the trajectory can be computed. Combining contact variables for the phase into a vector $\mathbf{c}$, the solution vector $\mathbf{s} \in \mathbb{R}^{(12(N+1)+N)K}$ that is optimized is

$$\mathbf{s} = \begin{bmatrix} \mathbf{x}_{1\ldots K} & \dot{\mathbf{x}}_{1\ldots K} & \mathbf{c}_{1\ldots K} \end{bmatrix}^T \tag{3.13}$$

The dynamics of my simple model correspond to those of a single rigid body with multiple forces acting on it from rectangular contact surfaces. In this setting, contact forces can efficiently be solved using either the approach of [**?**] or [**?**].

Chapter 4

CONTACT-INVARIANT OPTIMIZATION FOR HAND MANIPULATION

This chapter extends the contact-invariant optimization method presented in the previous chapter for automatic synthesis of dexterous hand movements, given only high-level goals specifying what should happen to the object being manipulated. Results are presented on a wide range of tasks including grasping and picking up objects, twirling them between the fingers, tossing and catching, drawing. The contribution here extends the unique contact model used in CIO which was specific to locomotion tasks, as well as applying the extended method systematically to hand manipulation.

4.1 Extension of CIO for Hand Manipulation

The original CIO method assumes that contact forces originate in rectangular patches on the body (such as soles of feet and palms of hand). Since contact force regions must be expressed as linear constraints in an inverse dynamics quadratic program (QP), they cannot support curved contact region geometry. While not too restrictive for full-body motion, this assumption is problematic when applying CIO to hand manipulation, because the surfaces of the fingers and palm are curved and hand motion often takes advantage of this (e.g. when rolling objects on the side of the finger).

To remove this limitation, in this work I include contact forces and their origin points as extra

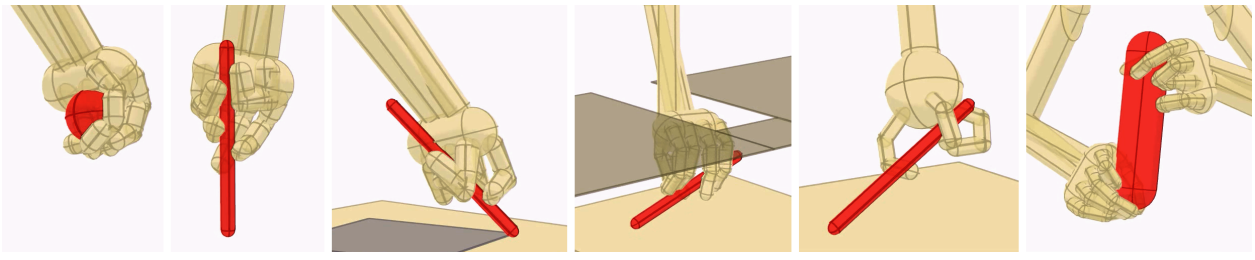


Figure 4.1: A selection of grasps and motions synthesized by the method in this chapter.

variables in the main optimization problem. I can then impose force origins to lie on arbitrarily-shaped regions through soft constraints. Friction force constraints can also be enforced on exact cone geometry, rather than pyramidal approximations of the cone. This reformulation removes the need to set up and use a quadratic programming solver to calculate the forces, which makes individual objective function evaluations much faster and makes the method easier to implement. The downside is that the size of the optimization problem becomes significantly larger. Nonetheless, I have found this approach to have optimization times similar to those reported in the original CIO work. Additionally, the gradient of the objective function (described in section 6.3.3) wrt the contact forces and their origins can be calculated analytically and does not require expensive finite difference calculation. I will now describe my CIO-based method specific to hand manipulation.

4.2 State and Trajectory

Given an initial state $\mathbf{s}_0$ of the entire system and a high-level goal (such as terminal object location), my method computes a state trajectory $\mathbf{s}(t)$ for a fixed period of time $[0, T]$. For simplicity, consider a system with one hand H and one object O to be manipulated. The state of the hand at any point in time is described as:

$$\mathbf{s}_H = \begin{bmatrix} \mathbf{x}_H & \dot{\mathbf{x}}_H & \mathbf{p}_i & \dot{\mathbf{p}}_i \end{bmatrix} \tag{4.1}$$

Where $\mathbf{x}_H \in R^6$ are the position and orientation of the palm, and $\mathbf{p}_i$ is Cartesian fingertip location of finger $i \in \{1...N_{\text{fingers}}\}$. Orientation is expressed with the exponential map [38] because of its suitability in trajectory optimization. From fingertip and palm locations, I can recover finger and arm joint angles and transforms using inverse kinematics (which in my case can be solved analytically). I assume the shoulder is fixed. The state of the object O at any point in time is described as:

$$\mathbf{s}_O = \begin{bmatrix} \mathbf{x}_O & \dot{\mathbf{x}}_O & \mathbf{f}_j & \mathbf{r}_j & c_j \end{bmatrix} \tag{4.2}$$

Where $\mathbf{x}_O$ are the position and orientation of the object. $\mathbf{f}_j$ and $\mathbf{r}_j$ and the force direction and origin point for contact $j \in \{1, ..., N_{\text{contacts}}\}$. $\mathbf{r}_j$ is expressed in the object's local coordinate system because simple trajectories in object space might trace complex trajectories in world space. $c_j \in$

$[0, 1]$ is an auxiliary variable that specifies if contact j is activite. In contrast with the original CIO formulation where c_j were unbounded, here I treat c_j informally as the probability of being in contact.

N_{contacts} is a fixed number of possible contacting bodies that can affect the object. The bodies I consider are:

- N_{fingers} contacts with the fingers, anywhere on the surface of the distal segments;
- one contact anywhere on the palm surface for each hand;
- two contacts anywhere on the ground surface.

Thus, $N_{\text{contacts}} = N_{\text{hands}}(N_{\text{fingers}} + 1) + 2$. See figure 4.2 for a visual description of the relevant quantities.

The entire system state is $\mathbf{s} = \begin{bmatrix} \mathbf{s}_H & \mathbf{s}_O \end{bmatrix}$ and the optimization problem variables are:

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}_k \end{bmatrix} \tag{4.3}$$

Where $k \in \{1, ..., N_{\text{keyframes}}\}$ are the discrete indices for the sequence of states being optimized. The first pose $\mathbf{s}_0$ (initial state) is specified by the user and is not part of the optimization variables. The motion trajectory is formed by interpolating these key states, spaced apart every 0.5sec in time. A motion phase is a segment between any two key states. At any continuous point in time t, the state $\mathbf{s}(t)$ is given by cubic spline interpolation over quantities $\mathbf{x}$, $\mathbf{p}$, linear interpolation for $\mathbf{f}$, $\mathbf{r}$, and constant interpolation for c. This choice of interpolation is motivated by how much complexity I expect in the trajectories for these quantities. Kinematic positions and orientations need complex spline trajectories (and thus need to include their velocities as optimization variables). Forces are interpolated linearly, while the auxiliary contact variables c are constant within each phase (which is how I defined the notion of phase).

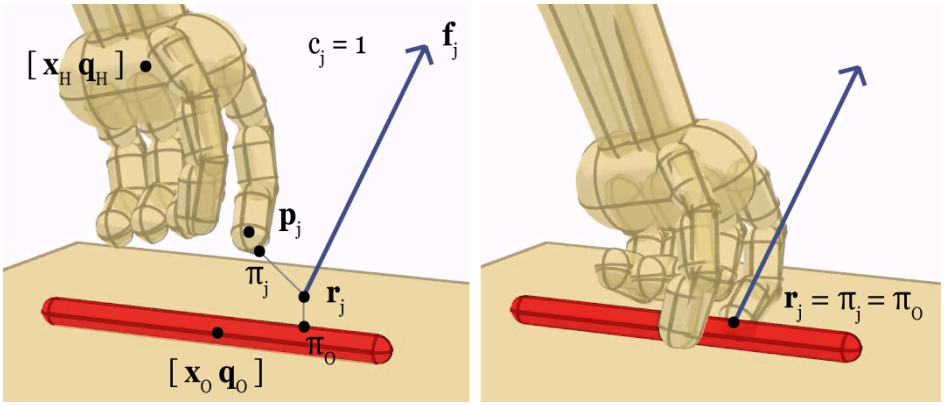


Figure 4.2: *Scene Model.* The quantities used in hand and object state. The left state is infeasible because contact c_j is active, but force origin $\mathbf{r}_j$ do not coincide with the hand and object. The state on the right is feasible.

4.3 Objective Terms

4.3.1 Contact-Invariant Cost

When contact j is active, the object and contact surface j must be coincident with each other, and the contact force origin $\mathbf{r}_j$ must lie on this coincident surface. The present version of the method is also restricted to motions with no slipping, so relative velocity at $\mathbf{r}_j$ must be zero. These constraints are satisfied when the following residual is minimized:

$$L_{\text{CI}}(\mathbf{s}) = \sum_j c_j(|\mathbf{e}_j^O|^2 + |\dot{\mathbf{e}}_j^O|^2 + |\mathbf{e}_j^H|^2 + |\dot{\mathbf{e}}_j^H|^2) \tag{4.4}$$

$$\mathbf{e}_j^O = \pi_O(\mathbf{r}_j) - \mathbf{r}_j \tag{4.5}$$

$$\mathbf{e}_j^H = \pi_j(\mathbf{r}_j) - \mathbf{r}_j \tag{4.6}$$

Where $\pi_O(\mathbf{r})$ and $\pi_j(\mathbf{r})$ are projections of on the surface of the object and of contact body j, respectively. I use capsules for all contact bodies, and as a result the projections are continuous functions of $\mathbf{r}$ and are fast to compute.

Note that this formulation is able to support arbitrary contact region geometry. For any point on the contact region, the above constraint will hold. For example, in the case of rectangular foot

contacting the ground, any $\mathbf{r}_j$ inside the contact rectangle will set (4.4) to zero. However the present approach is not limited to rectangular patches.

4.3.2 Physics Violation Cost

The hand H is connected to a fixed shoulder base, which makes the hand system fully actuated. Thus I assume that all kinematically feasible poses are also dynamically feasible, and place no dynamics constraints on the motion of the hand.

Since the object O is unactuated, the only way it can move is through application of contact forces. At any point in time, I place traditional Newton-Euler constraints on the motion of the object with the following residual:

$$L_{\text{physics}}(\mathbf{s}) = ||\mathbf{f}_{\text{tot}} - \dot{\mathbf{P}}||^2 + ||_{\text{tot}} - \dot{\mathbf{L}}||^2 + \lambda \sum_j ||\mathbf{f}_j||^2 \tag{4.7}$$

$$\mathbf{f}_{\text{tot}} = \sum_j c_j \mathbf{f}_j + \mathbf{f}_{\text{ext}} \tag{4.8}$$

$$_{\text{tot}} = \sum_j c_j \mathbf{f}_j \times (\mathbf{r}_j - \mathbf{x}_O) +_{\text{ext}} \tag{4.9}$$

Where $\dot{\mathbf{P}}(\mathbf{x}, \ddot{\mathbf{x}})$ and $\dot{\mathbf{L}}(\mathbf{x}, \ddot{\mathbf{x}})$ are change in linear and angular momentum of the object. $\mathbf{f}_{\text{ext}}$ and $_{\text{ext}}$ are any external forces and moments applied to the object including gravity. λ is a force regularization constant.

Note that while $\mathbf{f}_j$ follows a continuous trajectory, it is possible to model contact force discontinuities using the product $c_j \mathbf{f}_j$ and the fact that c_j is piecewise constant.

Additionally, I must constrain $\mathbf{f}_j$ to lie in the friction cone of the contact surface j. I use the contact surface normal at $\pi_j(\mathbf{r}_j)$ to define the cone normal $\mathbf{n}_j$. Then the angle between $\mathbf{f}_j$ and $\mathbf{n}_j$ must be less than the coefficient of friction μ. This is imposed as a half-quadratic soft cost L_{cone}.

4.3.3 Kinematic Violation and Task Cost

I place several kinematic constraints on the hand configuration. First, realistic limits on the finger and arm joint angles are imposed (recall that joint angles are available from analytical inverse kine-

matics). Second, because fingertip locations are independent optimization variables, their distance from the finger base must be restricted not to exceed the maximum length of the finger. Lastly, all collisions are penalized. Because all geometry consists of capsules, I can efficiently calculate and enforce a minimum distance between capsules. All the above constraints are imposed as soft half-quadratic costs and added up to form the cost term $L_{\text{kinematic}}$.

Humans prefer to use the bottom pads of their fingers for grasping objects. This is probably due to the higher friction in that skin area, as well as soft-issue deformations which create larger contact regions. Since I currently do not model these factors, I simply add an additional cost term L_{pad} which encourages $\mathbf{r}_j$ for a finger contact to be close to the distal bottom pad of the finger.

The task cost is a set of simple high-level goals ℓ_b that are behavior-specific and are provided by the user of my method. These task costs are constructed by selecting features of interest $\mathbf{h}_b$, providing their target values $\mathbf{h}^\star_{b,t}$ at certain points in time $t \in T_b$, and penalizing the difference between the actual and target feature values. Additionally, I place a small preference on smooth motion by penalizing hand and object accelerations.

$$L_{\text{task}}(\mathbf{s}, t) = \sum_b \ell_b(\mathbf{s}, t) + \lambda(||\ddot{\mathbf{x}}_O||^2 + ||\ddot{\mathbf{x}}_H||^2) \tag{4.10}$$

$$\ell_b(\mathbf{s}, t) = I\left[t \in T_b\right] \left\|\mathbf{h}_b(\mathbf{s}) - \mathbf{h}^\star_{b,t}\right\|^2 \tag{4.11}$$

Where $I[\cdot]$ is an indicator function for t matching one of the times when the goal is specified. For example, to create a task ℓ_{location} that commands the object location at the end of the movement, set $T_b = \{T\}$, $\mathbf{h}_b(\mathbf{s}) = \mathbf{x}_O$ and $\mathbf{h}^\star_{b,T}$ to the desired object position and orientation.

The typical weights on the costs I used are summarized in table 1. I have not found the algorithm to be particularly sensitive to the choice of these weights.

4.4 Numerical Optimization

I solve a box-constrained optimization problem to obtain $\mathbf{S}^\star$ described in (4.3), from which I can then obtain a continuous motion trajectory as described in section 4.2. The optimization problem

Table 4.1: Cost Term Weights

Cost Term	Weight	Cost Term	Weight
CI	10^1	task	$10^{-1 \text{ to } +1}$
physics/cone	10^0	pad	10^{-3}
kinematic	10^0	λ	10^{-3}

is:

$$\begin{aligned} \mathbf{S}^\star = \quad & \arg\min_{\mathbf{S}} \sum_t \sum_i L_i(\mathbf{s}(t)) \\ & c_j \in [0,1] \end{aligned} \tag{4.12}$$

Where i refers to the cost terms described in the previous section and t is a time index that subsamples the entire trajectory (in my case, I place t every 0.05sec of the trajectory). Thus, even though I optimize the motion at a very coarse time scale (key states every 0.5sec), I check for dynamic and kinematic validity at a fine scale.

I use a standard off-the-shelf box-constrained LBFGS algorithm [14] to solve (4.12). I use between 10 and 20 motion phases. The dimensionality of the problem is $N_{\text{phases}}(N_{\text{hands}}(12 + 6N_{\text{fingers}}) + N_{\text{objects}}(12 + 7N_{\text{contacts}}))$ which is from 1030 to 2060 in typical scenarios. Gradients of the objective function are computed with finite differencing (I use $\epsilon = 10^{-3}$). My implementation of finite differences takes advantage of the fact that variables affect the cost terms only at a few points in time and the costs for other times do not need to be recomputed.

The optimization is initialized with a static trajectory obtained by repeating the initial state for all times. This removes the need for hand-crafted initialization guesses specific to each motion. This initial trajectory is then perturbed with zero-mean Gaussian noise (with variance 10^{-2}) to break any potential symmetries. All contact variables c_j are initialized to 0.5 (which is the middle of their allowed range). For each example shown I run the optimization only once, i.e. there is no need to use a large number of restarts and eliminate all but a few of the solutions. The fact that I obtain good results in all cases suggests that even if there are local minima in this problem (which is very likely), most local minima correspond to successful motions. The alternative is that I am

extremely lucky, which I doubt.

I perform numerical optimization using continuation, in two stages. In the first stage, L_{physics} and L_{CI} are down-weighted to 0.1 of their typical values, which allows the optimization to explore potentially unphysical trajectories. In the second stage, all weights are at their weights specified in table 4.1. The solution at the end of the first stage is used to initialize the second stage. Despite the large problem size, I obtain optimization times of 2 to 6 minutes depending on the problem complexity. The optimizer executed between 500 and 1500 iterations in each stage before reaching a local minimum (or slowing down to a point that is hard to distinguish from a local minimum).

4.5 Results

By using simple task specifications ℓ_b or changing the environment, I can generate a wide range of hand/object interactions.

Grasping By using only ℓ_{location} (described in section 4.3.3) which specifies the terminal location of the object, my method automatically synthesizes entire hand and object trajectories. The synthesized grasps vary depending on object shape, e.g. a pen-like object leads to a different grasp compared to a ball-like object.

The type of grasp also varies depending on the specified final orientation. For horizontal orientation, the hand simply grips the object. For vertical orientation, a more complex grasp between fingers occurs in order not to twist the hand too much (and thus violate the joint limits). The final orientation can also be left unspecified. When the environment has obstacles, the hand will manipulate the orientation of the object to avoid the obstacles.

The resulting grasps are robust to external force perturbations. I show this by applying random forces of 0.25 Newtons at the object's tip in the second half of the trajectory.

Cyclic In-Hand Manipulation By connecting the last motion phase to the first phase, I can generate cyclic motions without any pre-specified initial state. By removing the ground environment (and its contacts), the object must be dexterously manipulated using only the hand. To synthesize cyclic object twirling motions, I use a task that commands a target orientation trajectory for the object and constant position for the hand. The target orientation trajectory is simply a full

constant-velocity rotation about one of the standard axes. The weight on L_{task} cost is low (10^{-1}) so the hand and object can deviate from this trajectory and form a natural gait.

Common Manipulation Tasks I generated a small sample of common hand manipulation tasks, such as drawing and dialing a cellphone. For the task of drawing a square, I specified that the tip of the pen object must reach each of the four corners of the square at every other motion phase. The need for the hand to pick up the pen was not specified and emerged automatically. Due to interaction of the pen with the the arm, the drawing task formed interesting trajectories that were not simple straight lines.

For the dialing task, constant object position and orientation were specified for the second half of the trajectory, and at every other motion phase the thumb fingertip location was specified to coincide with one of the buttons on the phone object.

Non-Humanoid Hand Morphologies The method is able to generalize to different hand morphologies, such as a three-finger hand similar to the Barrett Hand robot. I demonstrate a range of grasps and cyclic manipulation on this modified model. While the movement strategies are now quite different from a human hand, the high-level tasks are still accomplished.

Two-Handed Interaction The method is also able to generalize to a varying number of hands, with the hands cooperating to achieve the tasks and to evenly distribute the load (which is because of the force penalty term in (4.9)). I demonstrate grasping motions as well as a two-handed cyclic juggling motion. For juggling, a new ℓ_{contact} task was introduced that allowed only the contact variables c for the first hand to be active in the first half of the trajectory, and only the c for the second hand to be active in the second half of the trajectory. For one motion phase in the middle, the c's for both hands were penalized, restricting the object to be in the air for that phase.

4.6 Discussion and Future Work

In this chapter, I presented a method that can synthesize a wide variety of dexterous hand manipulation motions from only a few high-level goals on the object or hand, without any motion capture or detailed reference trajectory data. The method is not specific to human hands and can generalize to other morphologies. This was achieved by adapting the Contact-Invariant Optimization

(CIO) method to hand manipulation, and exploiting its ability to perform continuous optimization through contact events – which are numerous and exhibit complex structure in hand manipulation.

I was able to lift the previous restriction in the CIO method of contact regions being rectangular patches, and instead allowed contacts on curved surfaces. Another modification I made was to explicitly include contact forces and their origins as optimization variables. Note that these two changes are actually orthogonal. The first one is needed to apply CIO to hand manipulation. The second one was implemented because my brief preliminary testing indicated that it speeds up convergence, however I have not yet performed a systematic comparison. In other words, I do not yet know if it is better to optimize contact forces and their origins directly (while imposing physical realism with soft costs) or define them implicitly via inverse dynamics. In the latter case physical realism is automatically imposed and the number of variables being optimized is smaller, however the amount of computation per iteration increases (because I need an inverse contact solver) and furthermore an opportunity for continuation is lost. This comparison is a topic for future work.

Instead of using inverse contact dynamics, the new formulation only requires a projection function on the contact body. Although I only used single-capsule bodies, one could project onto more complex surfaces, or multi-capsule geometry. This would no longer tie contact to a single body, but instead could provide a floating "pool" of contacts to apply.

However, some restrictions present in the original CIO method still remain. By penalizing any relative velocity between hand and object (which prevents slipping contact), I cannot synthesize motions that purposefully slide the fingers along the object, or exploit slippage. Also, by using a coarse spline trajectory representation, I sometimes have difficulty with sharp motions such as hard collisions or separations. This is why some of my object motion may sometimes look "floaty". Either allowing discontinuities at spline knots, or having a second optimization at a finer timestep (initialized from the coarse result) may fix these issues. I also use a fixed number of phases and fixed phase durations as in the original CIO method.

Additionally, the dynamics and inertial effects of the hand are not taken into account in my method. Because the hand was not moving fast in my examples, I have not found this to be a problem, but modelling hand dynamics may be necessary in other cases.

4.6.1 Biomechanical and neural constraints

Some of the motions I synthesize were too rich, in the sense that the simulated hand moved in ways that a human hand could not. This is much less of an issue in synthesizing full-body movements, because even though most skeletal muscles act on multiple joints, the number of muscles and their attachment is such that the joints can be controlled largely independently. In contrast, hand muscles act through a complex network of tendons and the muscle-to-DOF ratio is smaller, introducing substantial coupling. This is not surprising given that morphologically the hand is very similar to a foot, and feet have evolved to distribute pressures rather than perform dexterous manipulation.

I should be able to increase the realism of the results by incorporating such constraints. This could be achieved using very detailed hand models [113] however it remains to be seen if such models are amenable to trajectory optimization. A different approach is to apply random forces to the tendons of simulated or physical hands – such as the Anatomically Correct Testbed robot hand [138] or cadaver hands [64] – and measure the correlations in the resulting movements. This statistical information can then be taken into account via additional cost terms. Similar information can of course be obtained from motion capture data, however it is likely to reflect not only biomechanical but also neural constraints – which may or may not be desirable depending on what I aim to accomplish. To avoid going back to motion capture databases that specify the movement details, one could extract the correlations common to a wide range of movement tasks, including basic tasks such as attemtping to move one joint at a time [124].

Chapter 5

CONTACT-INVARIANT OPTIMIZATION WITH MUSCULOTENDON ACTUATION FOR HUMAN MOTION SYNTHESIS

This chapter will present a trajectory optimization approach to animating human activities that are driven by the lower body. My approach is again based on contact-invariant optimization. I develop a simplified and generalized formulation of contact-invariant optimization that enables continuous optimization over contact timings. This formulation is applied to a fully physical humanoid model whose lower limbs are actuated by musculotendon units. The approach does not rely on prior motion data or on task-specific controllers. Motion is synthesized from first principles, given only a detailed physical model of the body and spacetime constraints. I demonstrate the approach on a variety of activities, such as walking, running, jumping, and kicking. The approach produces walking motions that quantitatively match ground-truth data, and predicts aspects of human gait initiation, incline walking, and locomotion in reduced gravity.

5.1 Humanoid Model and Forces

Our method is again an extension of contact-invariant optimization framework. The input is a set of sparse objectives, such as a target COM velocity for locomotion or a target foot position and velocity for a kick. The output is a kinematic trajectory for the humanoid model along with corresponding muscle activations, contact locations, ground reaction forces, and other parameters that describe the state of the simulated model over time. I provide an overview of both the model and the objective terms in this section.

The kinematic trajectory is a sequence of pose vectors over time. Each pose $\mathbf{q}$ contains 36 degrees of freedom (DOFs), including 6 for the root translation and orientation and 30 for the joints. I employ a model with mass distributions approximating a 180 cm, 70 kg male [137]. A

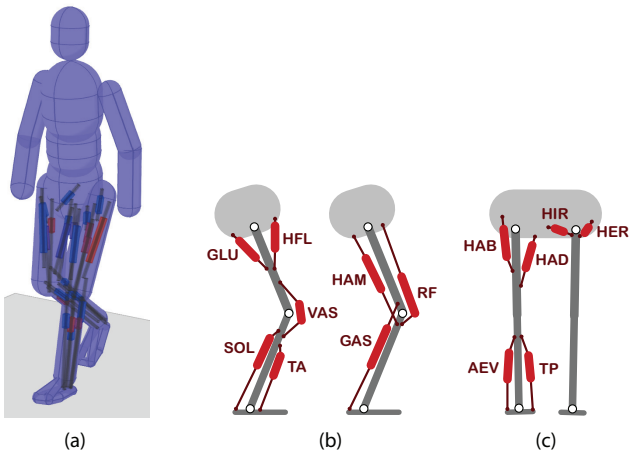


Figure 5.1: Humanoid model. (a) A visualization of the model during a walk. Red and blue cylinders in the legs correspond to the currently active and inactive MTUs. (b) Sagittal plane MTUs, defined following Wang et al. [137]. (c) Additional MTUs that provide control in the coronal plane (HAD, HAB, AEV, TP) and the transverse plane (HER, HIR).

physically consistent kinematic trajectory for the model is determined by external forces due to gravity and ground contact, as well as internal forces generated by the model. The internal forces actuate the humanoid model through torques generated at its 30 joint DOFs.

Our model of force generation for the lower body joints is motivated by properties of human musculotendon units (MTUs). Physiological properties of MTUs shape the torque patterns that can be produced by the human body. Given identical MTU control signals, the actual torque output at a particular joint can differ greatly depending on the current joint moment arms, muscle fiber length, and contraction velocity. Moreover, biarticular MTUs generate torques for pairs of joints, which leads to correlations in the torque patterns. Consequently, the model's capacity to generate a particular torque pattern is highly dependent on its kinematic state. I capture this dependency using Hill-type MTU models, which serve to increase the fidelity of the synthesized motion.

In particular, the lower body joints (hip, knee, and ankle) of my humanoid model are actuated exclusively by Hill-type MTUs. I use the MTUs that were used in the work of Wang et al. [137], as well as additional ones that support torque generation in the coronal and transverse planes. My model is illustrated in Figure 5.1. Four MTUs in each leg generate torque in the coronal plane:

the hip adductor and abductor muscle groups (HAD/HAB), the ankle evertor group (AEV), and the tibialis posterior (TP), which generates ankle inversion torques. I also add the hip internal and external rotation muscle groups (HIR/HER) that generate hip torque in the transverse plane. The choice of MTUs and their parameters are based on simplifying a more detailed musculoskeletal model [30].

While I do not model force generation in the upper body with MTUs, I treat active motor torques and passive torques separately. Passive torques provide a coarse approximation of the effects of musculoskeletal structure by applying spring-damper forces to the upper body joints as a function of kinematics [73, 134]. In particular, I assume that damping forces are applied to all upper-body joints, while the neck and back have weak tendencies to stay upright. The primary function of this treatment is to better model the effort of actively moving the upper body (Section 5.2.5).

Ground reaction forces are applied to the system when specific body parts come into contact with the ground. Since I am interested in capturing detailed pressure distributions over the feet, my model has six potential contact points on each foot, as illustrated in Figure 5.2. Two contact points are on the toe edge, two are on the ball of the foot, and two are on the heel. Ground reaction forces and foot locations are treated as free variables in the optimization. Their specific forms are described in Section 5.2.2.

When the motion of the system is physically consistent, the quantities described above are related by the equations of motion (EOM) for an articulated rigid body system:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{f}_{\text{GRF}} + \mathbf{f}_{\text{MTU}} + \mathbf{f}_{\text{passive}} + \mathbf{f}_{\text{motor}}. \tag{5.1}$$

Here M is the system inertia matrix, C is the matrix of Coriolis, centrifugal, and gravitational terms, and $\mathbf{f}_{\text{GRF}}$, $\mathbf{f}_{\text{MTU}}$, $\mathbf{f}_{\text{passive}}$, and $\mathbf{f}_{\text{motor}}$ are generalized forces due to ground reaction, musculotendon units, passive structures, and joint motors. The matrices M and C are determined by the mass distribution and the kinematics of the humanoid model. I discuss individual terms in the EOM in more detail in Section 5.2.

5.1.1 Objective Function

Since the input to my method is a set of sparse task objectives (J_{task}), a key requirement for my approach is to synthesize realistic movement for a variety of activities while keeping J_{task} as compact as possible. For example, the user only needs to change the target velocity to synthesize locomotion gaits appropriate for different speeds. Of course, in the absence of additional constraints, many trajectories $\mathbf{q}$ can satisfy J_{task}, including ones that are highly implausible. I must therefore use additional objectives that prioritize physical and biomechanical consistency. These *task-independent* objectives are universal for all motions I synthesize and include an EOM term, a contact consistency term, a body integrity term, and a musculotendon actuation term.

The EOM term (J_{EOM}) serves to enforce the equations of motion (7.6), which capture the requirement for gross dynamical consistency: bodies should move in ways that are consistent with their mass properties and forces in the system. The contact consistency term (J_{CIO}) governs the behavior of contact forces, so that forces are only applied when contact locations on the corresponding surfaces align. The body integrity term (J_{body}) enforces joint limits and prevents self-collision (e.g., legs going through each other). The musculotendon actuation term (J_{MTU}) requires control forces generated by musculotendon units to be consistent with muscle physiology. Finally, to resolve the remaining ambiguities, I assume a preference for economy of effort (J_{effort}). The complete objective is a linear combination of these terms:

$$E = \int_{t_i}^{t_f} \sum_k w_k J_k, \tag{5.2}$$

where $k \in \{\text{task}, \text{EOM}, \text{CIO}, \text{body}, \text{MTU}, \text{effort}\}$. The objective is in general nonconvex and I seek solutions through gradient-based optimization. The full state vector is defined in Section 5.3, which also describes the optimization algorithm. All the motions I synthesize use the same set of weights w_k.

Note that ideally all terms except for J_{effort} should be satisfied fully. Accordingly, I assign high weights to these terms so that they converge to near-zero values at the end of the optimization. In contrast, J_{effort} is inherently a soft constraint that disambiguates otherwise admissible motions that minimize all other terms. The effort term is particularly important for low-energy activities such as

walking: there are many physically and biomechanically consistent ways to move forward at low speed. For more energetic motions such as jumping and kicking, where the other terms sufficiently constrain the motion, J_{effort} does not influence the solution significantly.

5.2 Objective Terms

5.2.1 Equations of Motion

The synthesized trajectory is physically consistent if it respects the equations of motion (7.6). In practice, I compute all quantities in (7.6) except for the active motor torques $\mathbf{f}_{\text{motor}}$, which is recovered by rearranging the equation:

$$\mathbf{f}_{\text{motor}} = M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{f}_{\text{GRF}} - \mathbf{f}_{\text{MTU}} - \mathbf{f}_{\text{passive}}. \tag{5.3}$$

As discussed previously, only the upper body DOFs are actuated using joint motor torques, hence elements of $\mathbf{f}_{\text{motor}}$ that correspond to the root and lower body DOFs (denoted by Q_{root} and Q_{lower}) should vanish for physical consistency. I define the EOM term to drive the residuals for these DOFs towards zero:

$$J_{\text{EOM}} = \sum_i (f^i_{\text{motor}})^2, \tag{5.4}$$

where $q^i \in Q_{\text{root}} \cup Q_{\text{lower}}$. The matrices M and C are computed using MuJoCo [126] I discuss how the other generalized forces on the right hand side of (5.3) are computed in the rest of this section.

5.2.2 Contact Consistency

In principle, since $\mathbf{f}_{\text{GRF}}$ is a function of the body kinematics ($\mathbf{q}$) and contact geometry, one could naively employ a procedure that detects collisions and applies contact forces accordingly. In practice, however, such an approach results in intractable optimization problems, since hard dynamics constraints and contact discontinuities break the state space up into a jagged landscape with poor local minima. Traditional formulations of spacetime constraints avoid the issue by requiring user-specified fixed contact locations, which significantly restrict the range of motions that can be explored by the optimizer.

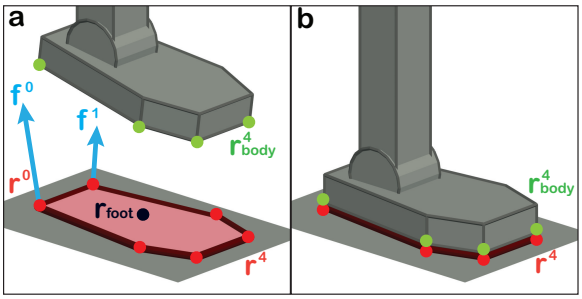


Figure 5.2: Contact-invariant optimization (CIO). Left: Although the contact points $\mathbf{r}^i_{\text{body}}$ on the foot are not coincident with the ground, CIO can still employ ground reaction forces $\mathbf{f}^i$ originating at points $\mathbf{r}^i$. Such "virtual" contacts are penalized by the contact consistency term J_{CIO}, but serve to smooth out the objective and assist the optimization. Right: A valid contact state is achieved when J_{CIO} is minimized.

I instead adopt contact-invariant optimization (CIO), which incorporates contact positions and forces into the optimization, and treats contact consistency as an objective term to be optimized alongside others. Figure 5.2 illustrates my treatment of contact positions and forces. Each foot has a virtual contact location with the ground at a planar world-space position $\mathbf{r}_{\text{foot}}$ and orientation θ_{foot}. Six individual contacts $\mathbf{r}^i$ are calculated relative to this virtual frame. The corresponding points on the body are denoted by $\mathbf{r}^i_{\text{body}}$. The ground applies a reaction force $\mathbf{f}^i$ at contact point $\mathbf{r}^i$. For each foot, the contact location $[\mathbf{r}_{\text{foot}} \;\; \theta_{\text{foot}}] \in \mathbb{R}^3$ and individual contact forces $\mathbf{f}^i \in \mathbb{R}^3$ are free variables in the optimization. Contacts contribute to (5.3) through the total contact force:

$$\mathbf{f}_{\text{GRF}} = \sum_i \mathcal{J}(\mathbf{q}, \mathbf{r}^i)^\top \mathbf{f}^i, \tag{5.5}$$

where $\mathcal{J}(\mathbf{q}, \mathbf{r}^i)$ is the Jacobian matrix mapping generalized velocities $\dot{\mathbf{q}}$ to contact force origin $\mathbf{r}^i$.

Three conditions should be satisfied for the combination of contact locations and forces to be physically plausible. First, the contact force should lie within the friction cone, which I enforce using bound constraints of the optimizer. Second, when a contact is not active, the contact force should be zero. Third, active contacts on the body and the ground should be touching and not sliding.

I now explain how the second condition is satisfied by the optimization. Consider a variable

$c^i \in [0, 1]$ whose value correlates with the validity and activity of contact i. Specifically, $c^i = 1$ if the contact is active and $c^i = 0$ if it is not. Unlike previous formulations of CIO, I do not maintain $\{c^i\}$ as separate free variables. Instead, I define them in terms of the normal contact force:

$$c^i = 0.5 \tanh(k_1 \|\mathbf{f}^i_\perp\| - k_2) + 0.5. \tag{5.6}$$

This forces the contact to deactivate when the normal force magnitude is zero. The parameters k_1 and k_2 govern the physical realism of the contact model. The closer (5.6) is to a step function, the more realistic the contact. However, if (5.6) is too sharp, the smoothing effect of CIO is reduced and the optimizer may fail to find a good solution. In practice, I first run the optimizer with $k_1 = 10, k_2 = 3$ and increase the sharpness to $k_1 = 20, k_2 = 2$ in a second optimization pass.

The function (5.6) is incorporated into the contact consistency term J_{CIO}, which also enforces the third condition:

$$J_{\text{CIO}} = c^i(\|\mathbf{r}^i - \mathbf{r}^i_{\text{body}}\|^2 + \|[\dot{\mathbf{r}}_{\text{foot}} \;\; \dot{\theta}_{\text{foot}}]\|^2). \tag{5.7}$$

The key idea of this formulation is that during the optimization contact points on the body may invoke ground reaction forces at a distance, through "virtual" contacts with points in the environment. That is, the optimization is allowed to explore intermediate solutions in which active "virtual" contacts ($c^i > 0, \mathbf{r}^i \neq \mathbf{r}^i_{\text{body}}$) generate non-zero ground reaction forces ($\mathbf{f}_{\text{GRF}}$) to reduce J_{EOM}. However, minimizing J_{CIO} minimizes such "virtual" ground reaction forces.

As mentioned above, I found it unnecessary to treat contact activations $\{c^i\}$ as additional variables in the optimization. Furthermore, by not explicitly modeling contact activation trajectories using piecewise constant functions, I remove the assumption that motions are broken up into discrete contact phases. Thus the precise timing of the contacts can be adjusted by the optimizer.

5.2.3 Musculotendon Actuation

Our Hill-type model is illustrated in Figure 5.3. It consists of a contractile element (CE), a serial-elastic element (SE), and a parallel-elastic element (PE). Let l_{CE} and l_{SE} denote the lengths of the elements connected in series. The total MTU length is then given by $l_{\text{MTU}} = l_{\text{CE}} + l_{\text{SE}}$. Conceptually, CE corresponds to the muscle fiber and SE corresponds to the tendon. Note that while the MTU

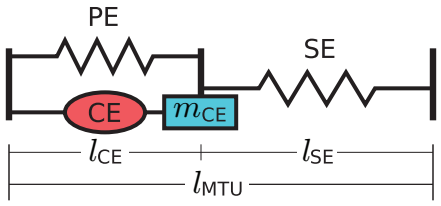


Figure 5.3: Hill-type musculotendon model. I employ an acceleration model [84] that regulates the acceleration of l_{CE} by introducing a point mass (m_{CE}) between the contractile (CE) and the serial-elastic (SE) elements.

length l_{MTU} is fully determined by the body pose $\mathbf{q}$, the length l_{CE} of the contractile element must be recovered separately and is included as a variable in the optimization.

The contractile element is responsible for generating active force F_{CE} given the control signal $a \in [0, 1]$:

$$F_{\text{CE}} = F_{\text{max}} f_l(l_{\text{CE}}) f_v(\dot{l}_{\text{CE}}) a, \tag{5.8}$$

where f_l, f_v are the force-length and force-velocity relations and F_{max} is an MTU-specific maximum isometric force parameter, which captures the strength of the muscle.

The serial-elastic element exerts a unidirectional nonlinear spring force that is a function of l_{SE}:

$$F_{\text{SE}} = F_{\text{max}} 0.5 \ln(1 + \exp(4(F'_{\text{SE}} - 0.5))), \tag{5.9}$$

where $F'_{\text{SE}} = 0.04^{-1}(l_{\text{SE}}/l_{\text{slack}} - 1)$, and l_{slack} is an MTU-specific tendon slack length constant. Note that (5.9) is a smooth approximation to the typical form: $F_{\text{SE}} = F_{\text{max}} \max(0, F'_{\text{SE}})^2$, where force generation can change abruptly (e.g., when the tendon becomes slack) and create difficulties for gradient-based optimization. The approximation is based on Tassa and Todorov [121].

Following Millard and Delp [84], the length of the contractile element l_{CE} is governed by its own set of equations of motion, which is derived by assuming that a point mass m_{CE} lies in between the contractile element and the tendon. The point mass (and therefore l_{CE}) is accelerated by the net force acting on it:

$$F_{\text{SE}} - F_{\text{CE}} - F_{\text{PE}} = m_{\text{CE}} \ddot{l}_{\text{CE}}, \tag{5.10}$$

where F_{PE} models the passive force generated by the MTU, which is also a function of the l_{CE}. I empirically set $m_{\text{CE}} = 0.1$.

Auxiliary variables introduced by the i-th MTU into the optimization are the control signal a^i and the length l^i_{CE} of the contractile element. The total force applied by the MTUs on the body is

$$\mathbf{f}_{\text{MTU}} = \sum_i \mathbf{m}_i(\mathbf{q}) F^i_{\text{SE}}, \tag{5.11}$$

where $\mathbf{m}_i$ is the moment arm vector of the MTU.

Our moment arms are computed as the derivative of corresponding MTU lengths (l_{MTU}) with respect to q [5]. The MTU length model is based on Geyer and Herr [36]. While effective for locomotion, their model for MTU-joint pairs with variable moment arms lead to unrealistic negative values for non-locomotion poses (e.g., knee extension angle in a crouch position). To prevent non-physical values while still capturing how MTU length changes with respect to the joint angles, I define

$$l_{\text{MTU}} = l_{\text{slack}} + l_{\text{opt}} + r_0 \rho (P(q - q_{\text{max}}) - P(q_{\text{ref}} - q_{\text{max}})), \tag{5.12}$$

where $P(x) = 2.0(1 + \exp(-2.1x))^{-1}$ is a scaled logistic function; l_{opt} is a MTU-specific optimal CE length; r_0 and ρ are MTU-specific attachment parameters; q_{ref} is the angle at which the l_{MTU} is in its default length, and q_{max} is the angle at which the moment arm is maximized.

The value of $\mathbf{f}_{\text{MTU}}$ is valid when Equation 5.10 is satisfied. I therefore define the musculotendon force objective in terms of the residuals of (5.10):

$$J_{\text{MTU}} = \sum_i (F^i_{\text{SE}} - F^i_{\text{CE}} - F^i_{\text{PE}} - m_{\text{CE}} \ddot{l}^i_{\text{CE}})^2. \tag{5.13}$$

The contraction dynamics model (5.10) accounts for the influence of the force generated by the MTU on the acceleration of the muscle mass. While the commonly used equilibrium models have been reported to be more efficient in forward simulations [85], I found that explicit modeling of acceleration is effective both for encouraging smooth l_{CE} trajectories and for guiding the optimization to biomechanically consistent solutions.

5.2.4 *Body Integrity*

Incorporating reasonable joint limits and preventing collisions between different segments of the body is clearly important for motion realism. Without a collision term, for example, the character can adopt gaits in which the legs pass through each other. I define

$$J_{\text{body}} = J_{\text{limit}} + J_{\text{collision}} \tag{5.14}$$

$$J_{\text{limit}} = \sum_i \{q^i - q^i_{\text{hlim}}\}^2_+ + \{q^i_{\text{llim}} - q^i\}^2_+ \tag{5.15}$$

$$J_{\text{collision}} = \sum_{a,b \in \mathcal{B}} \{\text{dist}(a,b) - (\text{rad}(a) + \text{rad}(b))\}^2_+, \tag{5.16}$$

where $\{x\}_+ = 0$ if $x < 0$ and $\{x\}_+ = x$ otherwise. q^i_{hlim} and q^i_{llim} are upper and lower joint limit values based on previous work [30, 137]. $\mathcal{B}$ is the set of rigid segments in my model, all of which are geometrically represented as capsules; $\text{dist}(a,b)$ is the shortest distance between the line segments representing the major axes of a and b, and $\text{rad}(a)$ is the radius of capsule a.

5.2.5 *Economy of Effort*

Our effort model consists of two components: a lower body term and an upper body term:

$$J_{\text{effort}} = J_{\text{lower}} + J_{\text{upper}}. \tag{5.17}$$

The effort J_{lower} exerted by the MTUs in the lower body is modeled using the metabolic energy expenditure model first presented by Anderson [8], which is the sum of heat released and mechanical work done by the MTUs. In particular, at every time step, the total rate of metabolic energy expenditure is given by

$$J_{\text{lower}} = \dot{A} + \dot{M} + \dot{S} + \dot{W}, \tag{5.18}$$

where $\dot{A}$ is the muscle activation heat rate, $\dot{M}$ is the muscle maintenance heat rate, $\dot{S}$ is the muscle shortening heat rate, and $\dot{W}$ is the positive mechanical work rate. The specific terms are defined in detail by Anderson [8].

The upper body term J_{upper} penalizes large active motor torques and accelerations generated by the upper body DOFs. As discussed in Section 5.2.1, active torques are obtained by rearranging the

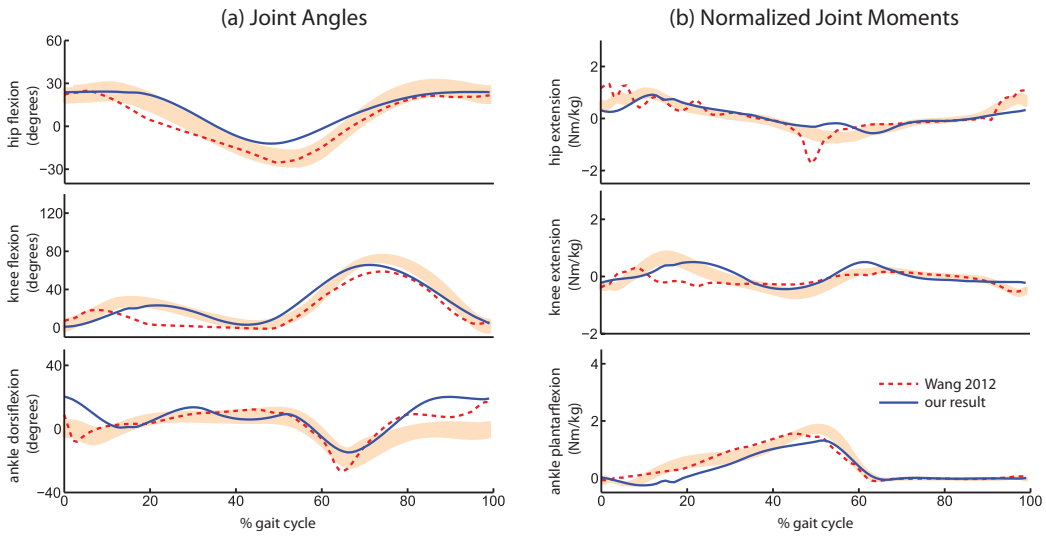


Figure 5.4: Locomotion at 1.5 m/s. The orange shaded areas represent one standard deviation of ground truth human data. Dashed lines represent walking data synthesized by the approach of Wang et al. [137]. Solid lines represent my approach. Although my approach does not assume a locomotion-specific control structure, it produces lower limb motion that is quantitatively closer to the ground truth in average standard error. In particular, my approach yields better predictions for the kinematics of the knee.

EOM after calculating other generalized forces in the system. I have already defined the contact force ($\mathbf{f}_{\text{GRF}}$) and the MTU force ($\mathbf{f}_{\text{MTU}}$). It remains to specify $\mathbf{f}_{\text{passive}}$: forces that capture passive effects of the musculoskeletal structure in the upper body. These are spring-damper forces for the set of upper body DOFs (Q_{upper}) [134]. These passive forces are designed to be a property of the humanoid model, and are task independent.

Let f^i_{passive} denote the i-th element of $\mathbf{f}_{\text{passive}}$. For $q^i \in Q_{\text{upper}}$, I define $f^i_{\text{passive}} = -k_p q^i - k_d \dot{q}^i$. For all other DOFs, $f^i_{\text{passive}} = 0$. Specifically, I set $k_p = 50, k_d = 1$ for the neck and back, capturing a weak tendency for the body to stay straight, and $k_p = 0, k_d = 1$ for all other upper body joints.

I can now recover $\mathbf{f}_{\text{motor}}$ through (5.3), and define the upper body effort term as follows:

$$J_{\text{upper}} = w_m \sum_i (f^i_{\text{motor}})^2 + \ddot{\mathbf{q}}^\top \mathbf{W}_a \ddot{\mathbf{q}}, \tag{5.19}$$

where $q^i \in Q_{\text{upper}}$. This penalizes large active torques and accelerations. $\mathbf{W}_a$ is a diagonal weight matrix, set to 10^{-5} for the neck DOFs, 10^{-4} for the back DOFs, and 10^{-6} for the rest of the DOFs in the upper body. Note the difference between J_{upper} and J_{EOM}: the upper body effort term encodes a preference for motions that require smaller active torques, while the EOM term approximates a hard constraint that there should not be any non-zero active torque in the root and lower body DOFs. Accordingly, I place a much higher weight on the EOM term.

5.3 Numerical Optimization

The full set of variables that are optimized for each time step is

$$\mathbf{s}_t = [\mathbf{q}_t \;\; \mathbf{r}_t \;\; \mathbf{f}_t \;\; \mathbf{a}_t \;\; \mathbf{l}_t],$$

which includes the generalized coordinate values of the humanoid ($\mathbf{q}_t$), contact locations on the ground ($\mathbf{r}_t = [\mathbf{r}^1_t, \ldots, \mathbf{r}^{10}_t]$), ground reaction forces ($\mathbf{f}_t = [\mathbf{f}^1_t, \ldots, \mathbf{f}^{10}_t]$), MTU activation signals ($\mathbf{a}_t \in \mathbb{R}^{28}$), and MTU contractile element lengths ($\mathbf{l}_t \in \mathbb{R}^{28}$).

By concatenating the state vectors and their time derivatives over M time steps, I obtain the vector $\mathbf{x} = [\mathbf{s}_1 \;\; \dot{\mathbf{s}}_1 \;\; \ldots \;\; \mathbf{s}_M \;\; \dot{\mathbf{s}}_M]$. The full trajectory is encoded with cubic splines, which serve to interpolate $\mathbf{x}$ and improve computational efficiency. At any time t, the state and its velocities and

accelerations can be calculated as $A_t\mathbf{x}$, where A_t is the spline encoding matrix. I thus numerically integrate the objective (5.2):

$$E(\mathbf{x}) = \sum_t \sum_k w_k J_k(A_t\mathbf{x}). \tag{5.20}$$

For all experiments, I set $w_{\text{task}} = 10$, $w_{\text{EOM}} = 10^2$, $w_{\text{CIO}} = 10^3$, $w_{\text{body}} = 10^3$, $w_{\text{MTU}} = 10$, $w_{\text{effort}} = 10^{-2.5}$, and discretize the integral at 30 Hz.

I use L-BFGS for optimization with numerically approximated gradients. I found the accuracy of the gradients obtained by standard finite differencing to be insufficient, causing premature termination. To obtain more accurate gradients, I use finite differences with complex arithmetic for the approximation [82]: $\frac{dJ_k}{d\mathbf{s}_n}(\mathbf{s}) \approx \frac{\text{Im}[J_k(\mathbf{s}+ih\mathbf{e}_n)]}{h}$, where h is an extremely small step size (I use 10^{-20}) and $\mathbf{e}_n$ is the nth standard basis vector.

I use seven spline knots per DOF for walking and running results, and 14 for jumping and kicking. The resulting dimensionality of the problem is over 2000, but the optimization does not require example-specific initialization. All examples are initialized to a default "zero" pose throughout the motion duration, where all elements of $\mathbf{q}_t, \mathbf{r}_t, \mathbf{f}_t, \mathbf{a}_t$ are set to zero. The MTU contractile element lengths ($\mathbf{l}_t$) are set to their respective optimal values (l_{opt}). From this initialization, I then proceed to run two optimization passes with different contact model settings. The first pass employs a smooth version of (5.6) with $k_1 = 10, k_2 = 3$. The result of the first pass is used to initialize a second pass with a sharper setting for (5.6): $k_1 = 20, k_2 = 2$. Both passes are run for a maximum of 3000 iterations.

Note that the objectives J_k are defined to be functions of the state vector at a single time instant, which means that the objective (5.20) and its gradient can be evaluated for different time steps in parallel. The independent nature of these terms enables rapid computation of numerical derivatives. As a result, despite the high dimensionality of the problem, all of the presented examples converge within 6 to 10 minutes on a machine with two Intel Xeon X5680 processors.

5.4 Results

5.4.1 Basic Locomotion

Locomotion is a particularly common human activity that is driven by the lower body. In the study of human locomotion gaits [94], the legs and pelvis are referred to as the "locomotor system" and the upper body is referred to as the "passenger unit." The passenger unit is "virtually a passive entity that is carried by the locomotor system"; it is also often referred to as the HAT (head, arms, and trunk/torso) to emphasize its secondary role in locomotion [94].

I synthesize locomotion by specifying task terms that capture the goal of moving the model's trunk forward with a target velocity and a target upright head orientation. Additionally, I require the motion to be periodic and symmetric. The periodicity constraint is implemented by requiring the last spline knot to be identical to the first. This constraint implicitly specifies the duration of the step. I select human-like durations given a desired target velocity based on relations between locomotion velocity and stride length derived from empirical observations [4]. The symmetry constraint restricts the variables associated with the left side of the body to be the same as the corresponding variables from the right side a period later.

Figure 5.4 compares the sagittal hip, knee, and ankle kinematics and joint moments of a 1.5 m/s walk synthesized by my approach to motion generated by a prior approach that relies on a task-specific control structure [137]. The results match the human ground truth more closely than the results of the prior approach, without reliance on a locomotion-specific control structure. (The joint angle and moment curves I compare to here are part of a dataset consisting of 20 healthy subjects walking and running at multiple speeds on an instrumented treadmill, see Wang et al [137] for details.) Specifically, I obtain an average standard error of 1.37 compared to 1.53 for [137]. Notably, my results better capture the knee trajectories. One likely reason is that the control structure assumed by Wang et al. was overly restrictive in defining the set of allowable control torque patterns. On the other hand, the hip kinematics indicate a lack of hip extension, which results in a shorter stride length relative to human data.

As demonstrated in the supplementary video, running gaits emerge when I specify larger ve-

locity and corresponding durations, without making any other modifications. However, I observe significant differences in knee and ankle angles during swing. In particular, my results exhibit both a lack of knee flexion and a slight over-plantarflexion during early swing phase. A closer look at the gait reveals that the tip of the foot model stays almost parallel over time and hugs the ground plane during the majority of the swing phase. A likely cause of this behavior is the lack of modeling of robustness and stability in my approach. Taking robustness into account would likely yield a gait with higher ground clearance and increased knee flexion during the swing phase, as observed in human running.

To evaluate the effect of musculoskeletal modeling and the fine-grained metabolic energy expenditure model, I replaced J_{lower} by the sum of squared torques of the lower body joints and optimized directly for joint torques in the lower body, without the use of MTUs. The weights on the squared torques were chosen so that the contribution of the effort term to the objective is comparable to J_{lower} for the MTU-driven model. The results are shown in the video. Walking synthesized by the purely torque-driven model suffers from severe crouching. The running gaits produced by the restricted model likewise suffer from dramatic artifacts.

The supplementary video also demonstrates walking results generated using the original CIO algorithm, which operates on a simplified character model with massless limbs. The original CIO formulation yields unnatural shuffling gaits in which the feet stay close to the ground. For running speeds, the original CIO algorithm fails to converge to reasonable solutions.

5.4.2 Locomotion Variations

Unlike the approach of Wang et al. [137], my approach is not limited to fixed-velocity locomotion. By modifying the environment and the simple task objectives, my approach can generate a variety of motions that are appropriate to different scenarios.

Gait initiation. To animate walking initiation, I specify an initial key pose (standing) at time zero, a final key pose at time 1.5 s, and linearly interpolate the target torso velocity from 0 to 1.5 m/s. The final key pose is set to one of the walking poses synthesized by my method. I

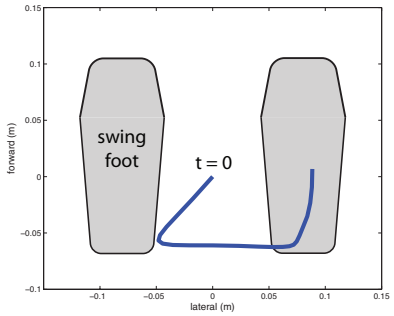


Figure 5.5: COP trajectory during simulated gait initiation. The shape of the trajectory matches human data.

found the method to be effective regardless of the precise choice of the walking pose. One salient characteristic of human gait initiation is the movement of the center of pressure (COP) [32]. As walking is initiated, COP moves backwards towards the heels and laterally towards the swing foot. Then, before the swing foot leaves the ground, the COP shifts laterally to the stance foot and moves from heel to toe as the step is taken. As shown in Figure 5.5, the motion generated by my approach exhibits this stereotypical COP trajectory. In the accompanying video, I demonstrate this walking initiation animation, as well as similarly generated running initiation.

Incline walking. By varying the slope of the ground, I can synthesize appropriate incline walking gaits. In the video, I demonstrate 1.5 m/s walking for different incline angles ($-40°$, $-15°$, $15°$, and $40°$). A notable feature of human walking on inclined surfaces is the maximum hip extension angle, which corresponds to how far the stance leg stretches behind the pelvis before it enters the swing phase. This angle is largest when I walk on level ground, and decreases for both upslope and downslope walking [66]. As seen in the video, the gaits synthesized by my approach reproduce these changes in lower limb kinematics. The peak ankle joint moments in the synthesized motions also increase as slope increases, as observed in human data [66].

Locomotion in reduced gravity. Gravitational acceleration has a direct effect on locomotion. It is well-known that a change in gravitational force, as on the Moon, leads to a change in human

locomotion gait. Notably, at typical terrestrial walking speeds, humans adopt gaits that are closer to running when gravity is reduced [15, 81]. My approach predicts this behavior. I set gravity to $0.16g$, as on the Moon, and synthesized locomotion at 1.5 m/s. As shown in the supplementary video, the synthesized gait resembles the gaits adopted by astronauts on the Moon.

5.4.3 Jumping and Kicking

Our approach can be used to animate a range of motions. I demonstrate its generality by synthesizing jumping and kicking motions. To animate jumping, I specify the initial and final poses (standing upright) and the trunk's height mid-jump. With these simple constraints, I synthesize forward, backward, and sideway jumps. As demonstrated in the video, features of human jumping such as knee flexion before lift-off emerge from the optimization. To animate kicking, I specify a target position and velocity for the ankle. As demonstrated in the video, a variety of different kicking motions can be generated by changing these simple constraints.

5.5 Discussion and Future Work

This chapter presented a trajectory optimization approach to animating human activities that are driven by the lower body. It demonstrated for the first time that a single optimization formulation can produce high-fidelity lower body motion for a range of human activities, without the need for prior motion data or task-specific control structures. This is achieved using a simplified and generalized formulation of CIO that optimizes over continuous contact timings and is applied directly to a detailed humanoid model in which the lower limbs are actuated by musculotendon units.

While effective, CIO does not completely prevent the optimization from converging to poor local minima. In particular, care must be taken to specify weights that trade off the importance of different terms in the objective. Visibly non-physical solutions that appear to "float" in the air can result from setting weights to EOM and CIO terms too low, while low weights on the MTU and effort terms tend to generate unnatural bent-knee gaits. Characterizing weights that can reliably generate a wide range of natural human motion is an interesting avenue for future work.

Like most methods based on spacetime constraints, the resulting control torques and activation signals generated by my work cannot be directly used in a forward simulation. This is both due to the spline encoding of the trajectories and the formulation of physical and biomechanical plausibility through soft constraints. The trajectories could serve to initialize more complex optimization schedules designed to generate strictly physical trajectories, which would be important for applications outside of animation as well as for synthesizing motions that further approach human ground truth.

The focus of this work was on the fidelity of the synthesized lower body motion. The movement of the upper body in my results does not match human ground truth in general. A likely reason is that my simple spring-damper model for upper body actuation does not properly represent the passive effects of the musculoskeletal structure in the upper body. A natural direction for future work is to develop a musculoskeletal model for the entire body, including the upper body. The highly detailed model developed by Lee et al. [67] provides a promising starting point, but some work is required to develop a simplified model that supports effective trajectory optimization. Such development would enable high-fidelity animation of activities that center on the upper body, such as reaching and pointing, as well as activities that require precise coordination of the entire body, such as ball sports, combat sports, and parkour.

Chapter 6

ENSEMBLE CONTACT-INVARIANT OPTIMIZATION THAT TRANSFERS TO PHYSICAL HUMANOIDS

While a lot of progress has recently been made in dynamic motion planning for physical humanoid robots, much of this work (including previous chapters in this thesis) has remained limited to simulation. In this chapter I show that executing the resulting trajectories on a physical robot Darwin-OP, even with local feedback laws, does not result in stable movements. I then develop a new trajectory optimization method, adapting contact-invariant optimization to plan through ensembles of perturbed models. This makes the resulting motion plans robust to model uncertainty, and leads to successful execution on the robot. I obtain a high rate of task completion without trajectory divergence (falling) in dynamic forward walking, sideways walking, and turning, and a similarly high success rate in getting up from the floor.

6.1 Darwin Robot

The Darwin-OP humanoid biped produced by Robotis Ltd. is a 26 degree-of-freedom robot – 3 root position, 3 root orientation, and 20 actuated joints. Each actuator is a MX-28 servo motor with position measurement of 12-bit resolution over 2π radians that is position controlled. An integrated inertial measurement unit in the robot's torso provides 3-axis acceleration and 3-axis angular velocity. Each foot also contains four force resistive sensors to measure contact forces. The sensor data output and control input are processed through an x86 CPU on-board the robot.

While the Darwin-OP is a convenient, low-cost biped platform, there are a number of limitations that must be overcome in one way or another. First, the control inputs to the servo motors are the set point for a PID controller. This means that the desired forces in the joints to achieve certain accelerations must go through this controller which I have no access to: each servo motor's

firmware is closed source. Secondly, while each body part of the Darwin robot can be disassembled and its mass measured, the center of mass of each body part and thus the entire robot is harder to predict. Although the dynamical model was initially derived from the CAD models of the robot which include inertia and center of mass measurements provided by Robotis, it is unknown if their model included all mass contributions such as the electronics boards, wiring, and hollow spaces in addition to the obvious metal frame and plastic body. While these are both sources of modelling error, I have tried my best to perform accurate system identification of these properties offline before trajectory optimization. System identification consisted of isolating motor parameters in a 1-DOF setup with known masses with mass measurements of each body component. The 1-DOF setup was given random control inputs with the motor positions and velocities measured as outputs; this data was used to optimize the motor parameters to reduce the error between actual and predicted states.

However, even with a perfect model, there are real world phenomena that the simulator does not take into account or cannot be identified. Geared motors frequently experience backlash where the gear teeth are not in contact with each other when reversing that need complicated estimation to identify [48]. This means that the joint can move while the servo reports the same position. This is in addition a programmed deadband, where the control logic of the servo will deliberately allow for some small error in its PID controller to prevent gears from wearing out. Together, these phenomena mean that idealized position controlled motors in a simulation can act differently from those in the real world unless the model is heavily augmented or specifically designed to compensate for these sources of error.

External factors also contribute to trajectory playback failure, namely contacts with the external environment. While they can be predicted and planned through, hard contacts are discontinuities in a position trajectory. This means that if a contact does not occur at the expected time, there is no assurance as to the rest of trajectory completing satisfactorily. Trajectories must then be planned in such a way to make up for missed or mistimed contacts. Furthermore, as contacts are the means through which a robot interacts with the world, they need to be made in an expected way to correctly position the robot's center of mass. Said another way, a biped robot needs to be

Figure 6.1: Simulated Darwin model (a), self-collision proxy (b), and environment contact points (c)

deliberate in its planning to stay balanced. I use model-based trajectory optimization to plan the motion of the robot.

6.1.1 Simulated Model

Our Darwin model includes all 26 degrees of freedom of the physical robot, denoted by $\mathbf{q} \in \mathbb{R}^{26}$. The model is based on CAD created specifications to manufacture Darwin. I have identified each actuator to have dry friction coefficient of 0.16 and damping of 1.13. The total mass of the robot is 2.835 kilograms. I assume contact between the robot and the ground to have friction coefficient of 0.75. For contact and actuator parameters, I use $k_c = 10$, $k_s = 36$ and $k_d = -12$ for all my simulations.

The Darwin robot geometry is a composition of complex, non-convex shapes which can make self-collision resolution a costly process and cause poor conditioning when used in trajectory optimization. To avoid these issues, I approximate the Darwin body with bounding capsules shown in figure 6.1c for the purposes of self-collision detection and resolution.

Our motions (especially such as robot lying on the ground or getting up) involve complex

interaction between Darwin and the environment. Feet, palms, knees, elbows, torso and head may be contacting the ground at any point in time. I predefine 23 potential contact points (shown in figure 6.1d), each of which can result in a 3-dimensional linear force being applied by the environment on the robot. I denote the total force vector to be $\mathbf{f} \in \mathbb{R}^{69}$.

The tasks I consider in this paper are centered around locomotion and getting and require robot standing and facing a particular orientation at the final time. This can be expressed as a quadratic cost c^T_{task} on final torso position and orientation, as well as final position and orientation of the feet.

6.2 Problem Description

6.2.1 Model and Actuation Dynamics

Let the model positions be denoted by $\mathbf{q}$. Physically consistent motions must satisfy the traditional equations of motion for an articulated rigid body system:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) = J(\mathbf{q})^T\mathbf{f} + \boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{r}) \tag{6.1}$$

Where M is the system inertia matrix, C is the vector of Coriolis, centrifugal, and gravitational terms, J is the contact force Jacobian. I use Mujoco physics engine [123] to calculate these quantities for any given state.

$\mathbf{f}$ is the combined vector of contact forces applied to the robot. I predefine a number of potential points $\mathbf{p}$ on the robot which will be making contact with the environment, each of which can result in a force being applied by the environment on the robot.

Contact forces and states must be consistent with each other. Contact forces must lie without the friction cone $\mathbf{f} \in C(\mathbf{q})$. If a contact is not active, the contact force should be zero. If a contact is active, the contact point should be touching the environment and not be sliding. The former can be written as a cone constraint, while the latter can be expressed as the following soft complementarity constraint for each contact i:

$$\lambda_i(\|\mathbf{p}_i - \text{proj}(\mathbf{p}_i)\|^2 + \|\dot{\mathbf{p}}_i\|^2) = 0 \tag{6.2}$$

Where proj() is the projection of a point onto the environment and λ is a smooth contact indicator variable calculated from contact normal force $\mathbf{f}^N$ as

$$\lambda = \frac{1}{2}\text{tanh}(k_c\mathbf{f}^N) + \frac{1}{2} \tag{6.3}$$

I term any state and its temporal derivatives satisfying the above constraints to be dynamically consistent.

I assume torques applied by the actuators to be a function of a PD controller driven by setpoints $\mathbf{r}$.

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{r}) = k_s(\mathbf{q}_{act} - \mathbf{r}) + k_d\dot{\mathbf{q}}_{act} \tag{6.4}$$

Where k_s and k_d are stiffness and damping coefficients and $\mathbf{q}_{act}$ are the degrees of freedom corresponding to the actuators.

6.2.2 Model Uncertainty

Even with identification, there is uncertainty in a number of model parameters, such as individual limb masses or their centers of mass. In addition, approximations I make, such as discrete number of contact points (instead of a contact surface) may make my simulation results diverge from reality. To diminish these issues, I take model uncertainty into account when performing trajectory optimization, as described in the next section. The sources of uncertainty I consider are:

Contact Point Location: As I use contact points defined as spheres instead of the convex mesh of the robot, I vary the location of the contact points. Not only does this allow for correction of poor sphere placement – a modelling error – the varying locations means that contacts can be made in unexpected ways and the trajectory can still succeed. For example, instead of a robot's foot needing to contact the ground specifically at one point on the heel, it has planned for contacts around the heel and even can be flat footed. Contact points are sampled from a Gaussian distribution centered around the nominal contact point location with a variance of 5mm.

Limb Mass: Although I and my collaborators took time during system identification to measure the mass of the robot's body parts, we cannot assume that a plan through the correct masses will succeed due to other source of error. By optimizing under various masses, optimized trajectories become more conservative in their movement. I vary masses by 20% of their nominal estimated mass.

Limb Center of Mass: There is also uncertainty in location of the center of mass. Similarly, trajectories are made more cautious and movements are more conservative. Center of mass locations are sampled from a Gaussian distribution centered around the nominal location with a variance of 20% of the limb's bounding volume.

I denote all model parameters by a vector θ. These model parameters affect terms M, C, J in equation (7.6).

In this work, I assume inaccuracies in motor-related parameters will result in short-horizon error and may be overcome at execution time either by built-in PD control, or feedback control in section 7.2.2. Thus, I do not explore uncertainty in the actuator parameters.

6.3 Deterministic and Ensemble Trajectory Optimization

6.3.1 Deterministic Case

Recall the deterministic contact-invariant optimization problem, which forms the base of the method in this chapter. Assuming deterministic model parameters θ, the problem is to find a dynamically and kinematically consistent trajectory of body configurations and contact forces $[\mathbf{q}\ \mathbf{f}]^{1:T}$ that achieves user-specified tasks. For clarity, I omit dependence on first and second time derivatives of the variables in the remainder of this paper.

$$\min_{[\mathbf{q}\ \mathbf{f}]^{1:T}} \frac{1}{T} \sum_{t} c^t(\mathbf{q}^t, \mathbf{f}^t) \tag{6.5}$$

This is a direct collocation approach to trajectory optimization, which directly optimizes state trajectories and dynamic and kinematic consistency is enforced as constraints (soft constraints in my case). I describe the exact objective function used in section 6.3.3.

6.3.2 Ensemble Case

I wish to find a robust behaviour which performs well under a variety of different model parameters θ. Taking a Monte-Carlo approach, I jointly optimize for a collection of N consistent state and contact force trajectories each of which satisfies task objectives. If there is no dependency between these trajectories under different instantiations of θ, the robot is free to behave completely differently under different models. I tie the trajectories together by requiring that they share a single PD setpoint trajectory, which will be used to control the physical robot.

$$\min_{[\mathbf{q}\ \mathbf{f}]_{1:N}^{1:T}} \frac{1}{N} \sum_{i,t} c^t(\mathbf{q}_i^t, \mathbf{f}_i^t) + \frac{1}{2} \left\| \mathbf{r}_i^t - \bar{\mathbf{r}}^t \right\|^2 \tag{6.6}$$

Where $\bar{\mathbf{r}} = \frac{1}{N} \sum_i \mathbf{r}_i$ is the average PD setpoint trajectory. $\mathbf{r}_i^t$ can be recovered at any point in time from equations (6.4) and (7.6).

$$\mathbf{r}(\mathbf{q}, \mathbf{f}) = \mathbf{q}_{act} + \frac{k_d}{k_s} \dot{\mathbf{q}}_{act} - \frac{1}{k_s}(M\ddot{\mathbf{q}} + C - J^T\mathbf{f}) \tag{6.7}$$

If $\mathbf{q}_i$ and $\mathbf{f}_i$ are interpreted as samples from their respective random variables, one way of interpreting (6.6) is to reduce the expected cost under random model parameters, while reducing the variance of the resulting PD setpoint trajectory.

6.3.3 Objective Function

Our objective is a weighted combination of terms, each depending only on state variables and its temporal derivatives at a particular point in time.

$$c^t(\mathbf{q}, \mathbf{f}) = \sum_k w_k c_k^t(\mathbf{q}, \mathbf{f}) \tag{6.8}$$

The terms I include are:

Equations of Motion and CIO Constraints: I require state and contact forces to satisfy dynamic consistency constraints as described in section 6.2.1. Equations (7.6) and (6.2) are enforced as soft constraints with quadratic costs.

Kinematic Constraints: I require that states $\mathbf{q}$ be kinematically consistent. Contact points should not be penetrating the environment, body parts should not be self-intersecting and joint angles must be within model limits. These bound constraints are enforced softly as half-quadratic costs.

Regularization: To get more natural trajectories and to improve problem conditioning, I prefer state and contact force trajectories to be smooth. Additionally, I prefer PD setpoint to not be too far from the state, which implicitly minimizes velocity and control effort.

$$c_{\text{reg}} = \|\ddot{\mathbf{q}}\|^2 + ||\ddot{\mathbf{f}}||^2 + \|\mathbf{q}_{\text{act}} - \mathbf{r}\|^2 \tag{6.9}$$

User Task: While the above terms put us in a space of plausible trajectories, the trajectory must also satisfy tasks specificed by the user. These objectives can be open-ended, and are described in section 6.1.1.

Cost Term	eom	cio	kin	reg	task
Weight	10^1	10^1	10^0	10^{-5}	10^0

Table 6.1: The weights for the terms contributing to the objective function (6.8)

6.3.4 Optimization Algorithm

To solve the above optimization problem, I use a trust region method with Levenberg-Marquardt heuristic. The method requires inversion of the objective Hessian, but in my case each c^t only depends on $\mathbf{q}^t$, $\mathbf{f}^t$ and its time derivatives. Thus, the Hessian is block-diagonal and can be inverted efficiently using Cholesky factorization.

I use $N = 10$ model samples, which is fairly small and allows us to perform optimization in (6.6) without additional approximations, such as stochastic or distributed optimization methods. I found the method to converge between 100 to 500 iterations, depending on complexity of the problem.

6.3.5 Feedback Control

In open loop control, I directly use $\bar{\mathbf{r}}$ as the PD setpoint trajectory sent to the robot. But as discussed in section 6.1, there are various modelling and system deficiencies that must be overcome for successful trajectory playback. Ensembles of certain parameters can overcome modelling errors, but feedback is necessary to overcome unmodeled issues, namely the limitations of the servo motors. For this reason, the sensory features chosen for feedback consisted of joint angles and joint angle velocities. The goal is to accurately track the trajectory's position and speed. Other sensory features were considered, and will be discussed in 6.6.

Let $\mathbf{x} = [\mathbf{q}\ \dot{\mathbf{q}}]$ be the state of the system in forward dynamics simulation. I first execute my PD setpoint trajectory $\bar{\mathbf{r}}$ in forward simulation under the average model parameters to get a rollout $\bar{\mathbf{x}}^{1:T}$. I linearize about this rollout and perform an LQG backward pass to compute the optimal time-varying feedback gains A^t. The resulting PD setpoint sent to the robot using feedback is:

$$\mathbf{r}_{\text{FB}}^t(\mathbf{x}) = \bar{\mathbf{r}}^t + A^t(\mathbf{x} - \bar{\mathbf{x}}^t) \tag{6.10}$$

Where $\mathbf{x}$ contains the joint angles and joint angle velocities coming from the physical sensors. Because I do not have access to root position and velocity components of $\mathbf{x}$, I set the corresponding entries of A to zero.

6.4 Experimental Methods

6.4.1 Trajectory Preparation

After an optimal trajectory is calculated, the first step in validating its success on a real robot is to test in simulation. I again use the MuJoCo physics engine to play back the calculated controls in forward dynamics. This serves two purposes. First, the researchers can quickly examine the optimized trajectory the potential of real world success. If the trajectory causes the robot to fall in the simulation, it is very likely that the Darwin robot would also fall.

The second purpose is that the forward dynamics simulation is used to find the reference trajectory used with feedback. In (6.10), there is the result of the equation, $\hat{\mathbf{q}}_{\text{darwin}}(\mathbf{s}, t)$, that is the

desired position after accounting for potential state error, and also $\mathbf{s}$, the ideal positions when there is no error. As a sequence of desired joint states is used during feedback gain calculation, and the Darwin robot's control input is the joint positions, this reference trajectory acts as the desired optimal trajectory when feedback is not used.

6.4.2 Trajectory Execution

To save computation time, the optimization and forward dynamics happen at a slower time-step than the robot's control loop. In order to control the Darwin with the trajectory at a slower time-step, I linearly interpolate between the optimization's time-steps to get the control input based on the Darwin's system time. This is important for feedback especially. A higher update rate means that state error should be corrected for before it grows too large. Furthermore, as modelling errors are always present, larger time-steps will encourage divergence from the planned trajectory as errors will grow with time.

At the start of every trajectory playback, the Darwin loads the trajectory data, with timing information, from file into memory. Then, it assumes the initial position and waits for input to begin the trajectory playback. When prompted, the Darwin notes the time: this because the start time of the trajectory. It immediately starts progressing through the trajectory data, interpolating as necessary to correctly match the time data of the provided trajectory.

6.4.3 Experiment Design

The reference trajectory is played back on the Darwin robot as the baseline result. This is the desired trajectory for a hardware platform that is identical to the model used in optimization. Ensemble optimization is used to find more robust trajectories which are then compared with the reference. Furthermore, for both ensemble and reference trajectories, I both use and do not use feedback. As feedback is another means of helping the Darwin complete a trajectory, I wish to isolate the effects.

For each optimized trajectory, I have four trials of 20 runs each. The four trials are with and

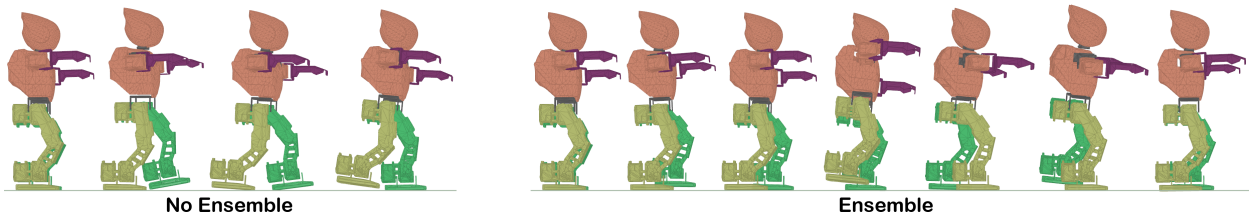


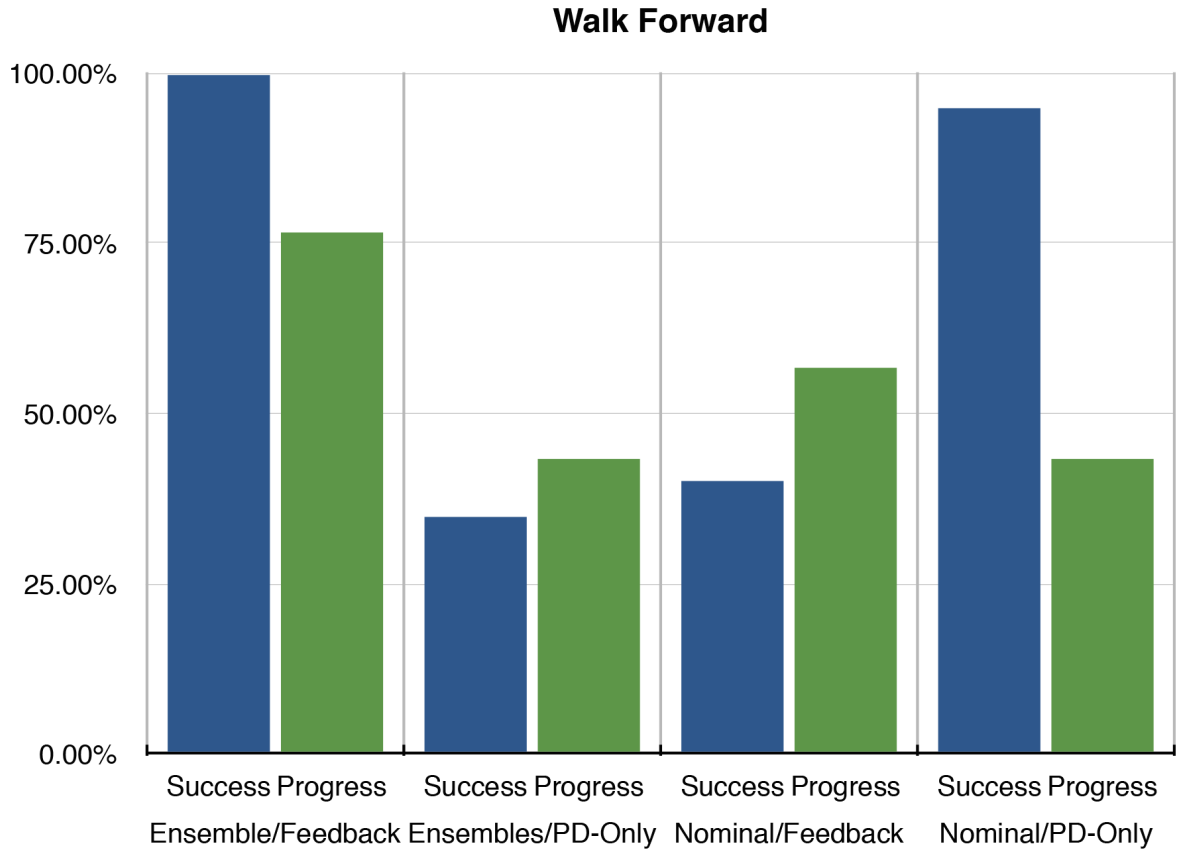
Figure 6.2: Differences between trajectories optimized without and with the ensemble method

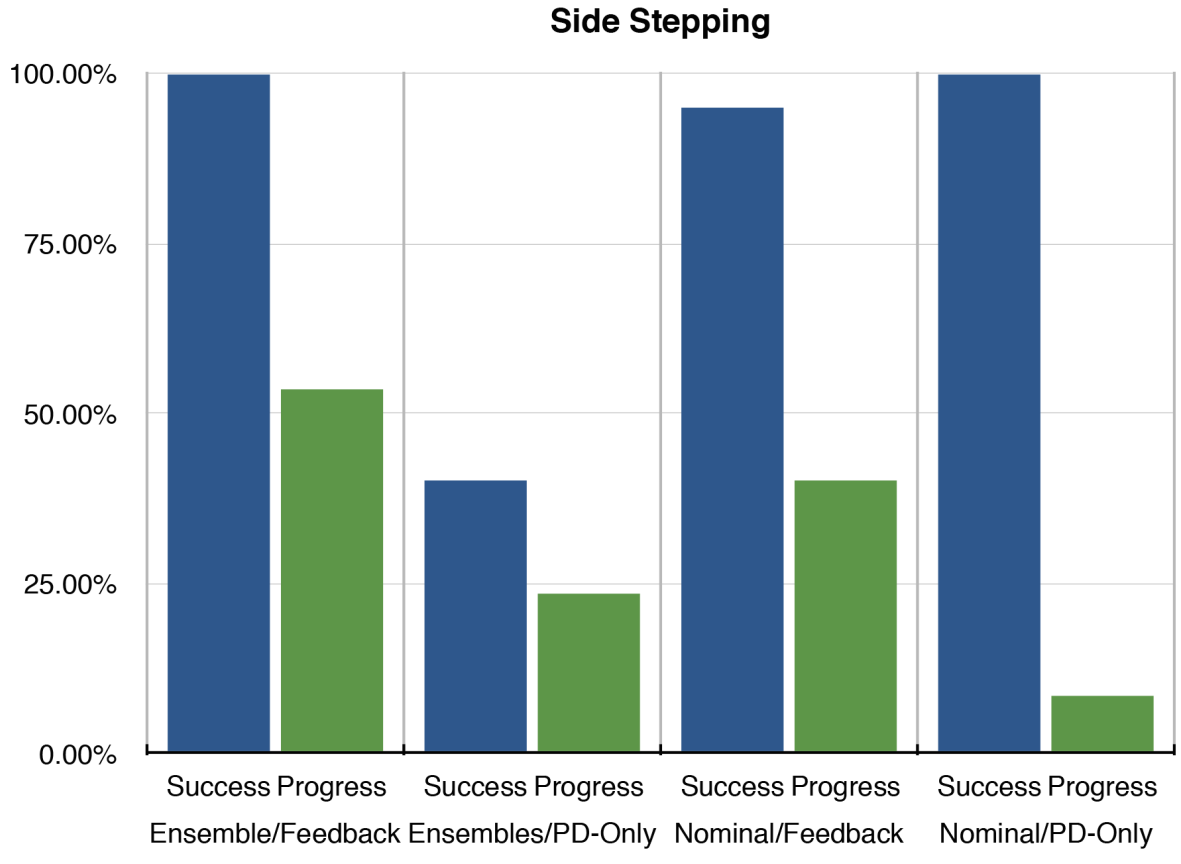
without ensemble optimization and with and without feedback used in the playback. The optimized trajectories are for the actions of forward walking 30 centimeters, a 90 degree turn to the right, and for side stepping 30 centimeters.

These were chosen as they only required contact between the robot and the ground, had simple measure of action success, and were dynamic enough that they could critically diverge from the planned trajectory. While a robust trajectory allows for some movement from the plan, falling is considered to be a sign of an unmitigated failure. Action success is determined by the robot not falling, while progress is the intended result of the trajectory, e.g. walking forward should allow the Darwin robot to move in the direction it is facing. This is measured by a Phasespace tracking system using 8 LEDs attached to the robot's torso that provide a rigid body position and orientation to overhead cameras. This data is processed after data collection to give us its translational and rotational progress. Progress in walking and side stepping is measured by left multiplying each translational position with the unit vector representing the total distance travelled, while in turning progress represents the Euler angle of travel accomplished.

6.5 Results

The results of the experiments are summarized in figure 6.3. In short, methods optimized without ensemble did not necessarily fall, but they also did not successfully complete their actions. By contrast, ensemble-optimized methods both stayed upright and successfully completed their actions. Another interesting observation is that while feedback improved the performance of ensemble-optimized methods, it actually induced more instability that would cause Darwin to fall for a non-


Walk Forward
100.00%
75.00%
50.00%
25.00%
0.00%
Success Progress Ensemble/Feedback
Success Progress Ensembles/PD-Only
Success Progress Nominal/Feedback
Success Progress Nominal/PD-Only


Side Stepping
100.00%
75.00%
50.00%
25.00%
0.00%
Success Progress Ensemble/Feedback
Success Progress Ensembles/PD-Only
Success Progress Nominal/Feedback
Success Progress Nominal/PD-Only

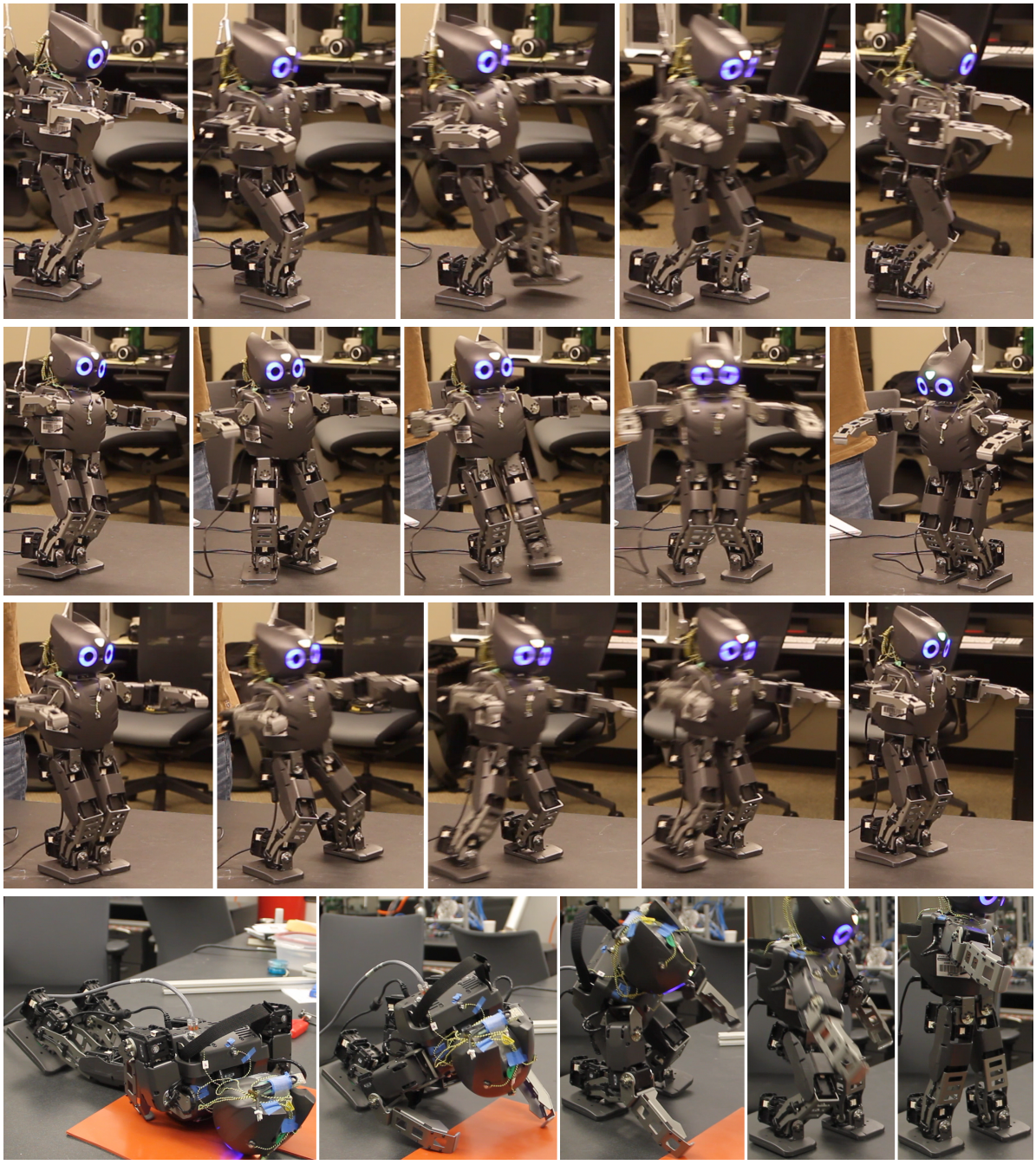


Figure 6.4: Results of successfully executing ensemble-optimized trajectories with feedback for walking, turning, sidestepping, and getting up

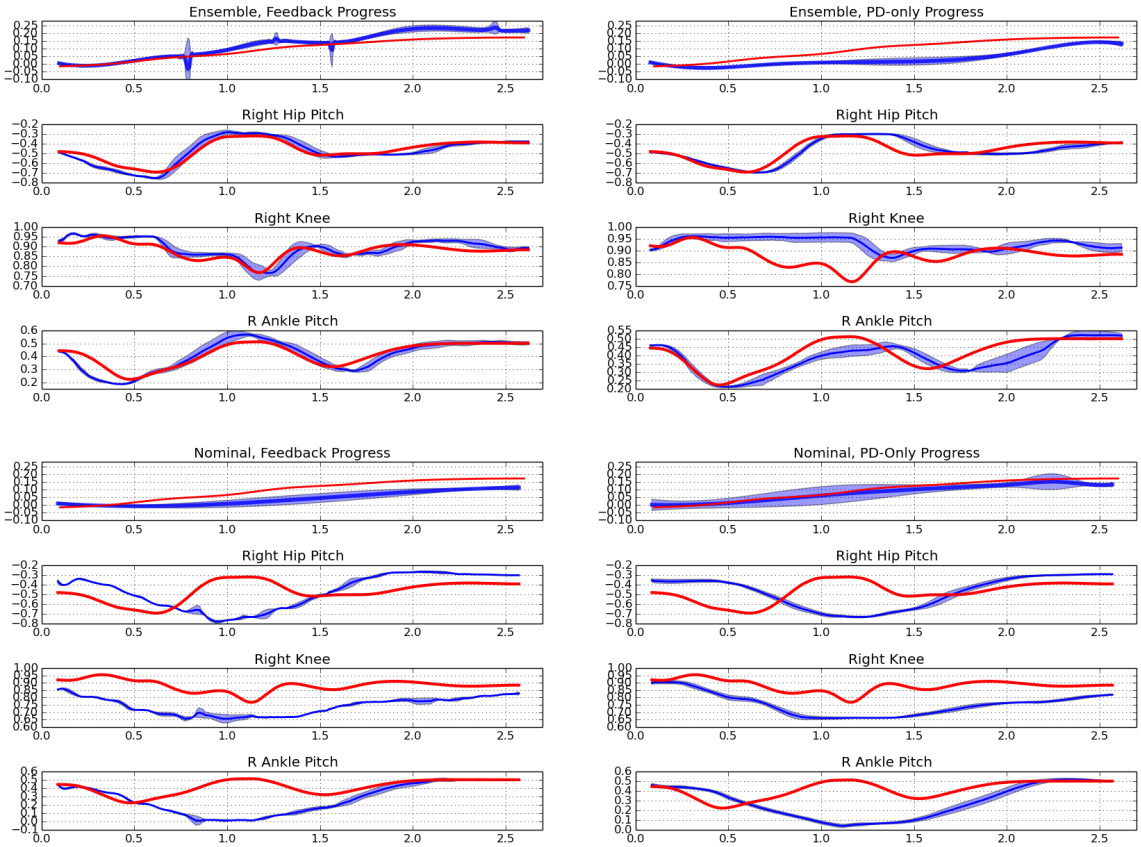


Figure 6.5: Mean and variance (in blue) compared to the reference position (in red) for walking forward. Progress was determined by calculating the robot's translational position between its starting and final positions. Three joint positions are displayed as well to highlight the effects of ensemble optimization and feedback. Without ensemble, the actual joint positions hardly tracks their control signals. With feedback (and ensemble), the same joints follow the reference much more closely. As a result, the progress made with both ensemble and feedback most closely reaches the target position.

ensemble method.

I see the qualitative differences between walking motions optimized with two different methods in figure 6.2. In particular, note that non-ensemble method lands on heel and transfers weight from heel to toe during double support phase. This is a natural, but precarious strategy. In contrast, the ensemble method lands with the foot flat on the ground. Non-ensemble method also swings the foot really close to the ground surface, whereas ensemble method raises to foot more during

swing phase. Non-ensemble method also takes a smaller number of large steps, whereas ensemble method takes a larger number of smaller steps. In general, the movement strategies found by ensemble method are more conservative, whereas non-ensemble strategies are riskier (although arguably closer to what humans may actually do).

6.6 Discussion and Future Work

The results demonstrate that while non-ensemble trajectories might be stable enough on the Darwin to not fall, they do not make progress appropriate for the given action. This means that for the walking, turning, and side-step actions on a flat ground surface, the contacts with the ground did not produce the appropriate torque to propel the robot in the correct direction. In effect, the robot was slipping and sliding in place. This suggests that either the expected friction of the ground surface was wrong, or the trajectory that used the frictional contact for torque was insufficient or incorrect. Regardless, the non-ensemble trajectories either need accurate models to plan through or can only be used for more simple trajectories.

Ensemble trajectories had a much higher chance of completing their respective action. While the Darwin did fall, the fact that trajectory progress was made indeed suggests that modelling errors contributes to poorly optimized trajectories. The resulting trajectories through ensemble optimization demonstrate more cautious behavior: the feet are lifted off the ground higher to clear ensemble contact points, and more stable trajectories are chosen that are dynamically consistent with the real robot.

Feedback had some interesting effects on the Darwin. When using feedback, the robot would frequently experience some form of overshoot when correcting for errors. This would obviously be problematic for trying to remain stable, and shows as feedback with non-ensemble trajectories were in fact prevented from successfully completing. The reason, I suspect, is that as both the trajectory and feedback were calculated from an incorrect model, errors built up more quickly, and could not be correctly compensated for.

Feedback with the ensemble trajectories did show improvement, however. Trajectories would

frequently reach 'tipping points' where a successful transfer of mass would need to take place to continue the trajectory – otherwise, the robot would fall. Feedback would help in these instances to maintain the correct velocity within the joints to move correctly past the tipping point.

While this work demonstrates the ability of my ensemble method to make optimized dynamic trajectories feasible for use on robotic hardware, there are more avenues to explore. First is the types of ensembles and how they pertain to different actions. Robots that rely on contact with their environment might need different parameters to be in the ensemble than robots that do not.

Secondly, the feedback policy's range of use seems to be narrow in this experiment. Strong perturbations from the trajectory cannot be recovered. How strong remains to be seen and measured. Furthermore, I wish to explore how increasing the state space of the feedback can improve the feedback policy or increase robustness. Instead of just joint positions and velocities, I could include the robot's root orientation and position, and associated velocities. I hypothesize that this would increase the degree of success of a given action – walking forward a specific distance would be more accurate to that specific distance as I would have a control gain on the error in the distance. Some sensors I consider using are the Darwin's internal IMU or foot force sensors, or an external Phasespace position and orientation tracking system.

Alternatively, the simple feedback policy could be replace with a neural network based policy. This could result in very different policies for different models, but all derived from the same parameters. Neural network policies could be used to increase robustness by expanding the state space from which a robot could recover from perturbations. This means, of course, measuring the window of robustness. This could be done in simulation, or by injecting noise into the sensor measurements or control outputs of the robot and again measuring the success of the action.

Chapter 7

SYNTHESIS OF INTERACTIVE CONTROLLERS WITH TRAJECTORY OPTIMIZATION AND NEURAL NETWORKS

Interactive real-time controllers that are capable of generating complex, stable and realistic movements have many potential applications, including animation, gaming and robotic control. They can also serve as computational models in biomechanics and neuroscience. Designing such controllers however is a time-consuming and largely manual process. My goal is to automate this process, by developing universal synthesis methods applicable to arbitrary behaviors, character morphologies, online changes in task objectives, perturbations due to noise and modeling errors. Key to my approach is the power of modern computers, which have become fast enough to search/optimize over the vast space of candidate controllers, and find one that works. It is of course impossible to overcome the curse of dimensionality in a brute-force way, and my method deploys a number of strategies to produce controllers for all characters and control tasks.

Every automated search starts with a choice of representation. Earlier approaches to interactive real-time control relied on interpolation in motion capture datasets, as well as state machines designed for specific tasks. In contrast, I seek a representation that does not require significant manual tailoring for application to new characters/tasks. One such representation would be explicit

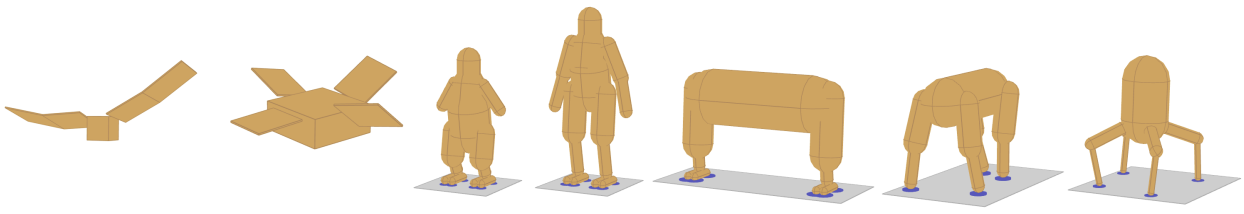


Figure 7.1: A sample of characters controlled interactively by method in this chapter.

trajectories of states or controls, possibly augmented with local feedback controllers. Complex movement trajectories, involving multiple contacts and other non-linearities, have recently been optimized automatically. However such optimization takes at least minutes of CPU time and yields local solutions. One general approach to overcoming this limitation is model-predictive control, where the trajectory is re-optimized as often as possible, using the previous solution for warm-start. This has given rise to impressive results, but is limited to behaviors where short-horizon planning (on the order of a second) is sufficient, and furthermore if the optimizer gets into a poor local minimum it is not clear how to proceed in real-time. Nevertheless, given that trajectory optimization is the only approach that has consistently produced behaviors with the complexity and realism I am aiming for, here I seek to leverage it for the purpose of interactive real-time control.

The representation I choose for my global controller is a neural network, with a control-specific architecture described below. Neural networks have been used in multiple fields for decades. They are currently experiencing a resurgence, thanks mostly to faster computers and larger datasets, and to some extent improved training methods. Even though control was historically among the earliest applications of neural networks, the recent success stories are in computer vision, speech recognition and other classification problems that arise in artificial intelligence and machine learning. In these domains large datasets are readily available on the Internet. In contrast, the data needed to learn neural network controllers is much harder to obtain, and in the case of imaginary characters and novel robots we have to synthesize the training data ourselves (via trajectory optimization). At the same time the learning task for the network is harder. This is because we need precise real-valued outputs as opposed to categorical outputs, and also because our network must operate not on i.i.d. samples, but in a closed loop, where errors can amplify over time and cause instabilities. This requires specialized training procedures where the dataset of trajectories and the network parameters are optimized together. Another challenge caused by limited datasets is the potential for over-fitting and poor generalization. My solution is to inject different forms of noise during training. The scale of the problem requires cloud computing and a GPU implementation, and training that takes on the order of hours. Interestingly, I invest more computing resources in generating the data than in learning from it. Thus the heavy lifting is done by the trajectory optimizer, and yet the

neural network complements it in a way that yields interactive real-time control.

The method in this chapter automatically generates neural network parameters that represent a control policy for physically consistent interactive character locomotion control, requiring a dynamical character model and task description. At the core of my approach is the merging of both trajectory optimization and stochastic neural networks together. Such coupling enables the trajectory optimizer to act as a teacher, gradually guiding the network towards better solutions. This combines correct behavior with real-time interactive use. Furthermore, the same algorithm and controller architecture provide interactive control for multiple creature morphologies.

7.1 Overview

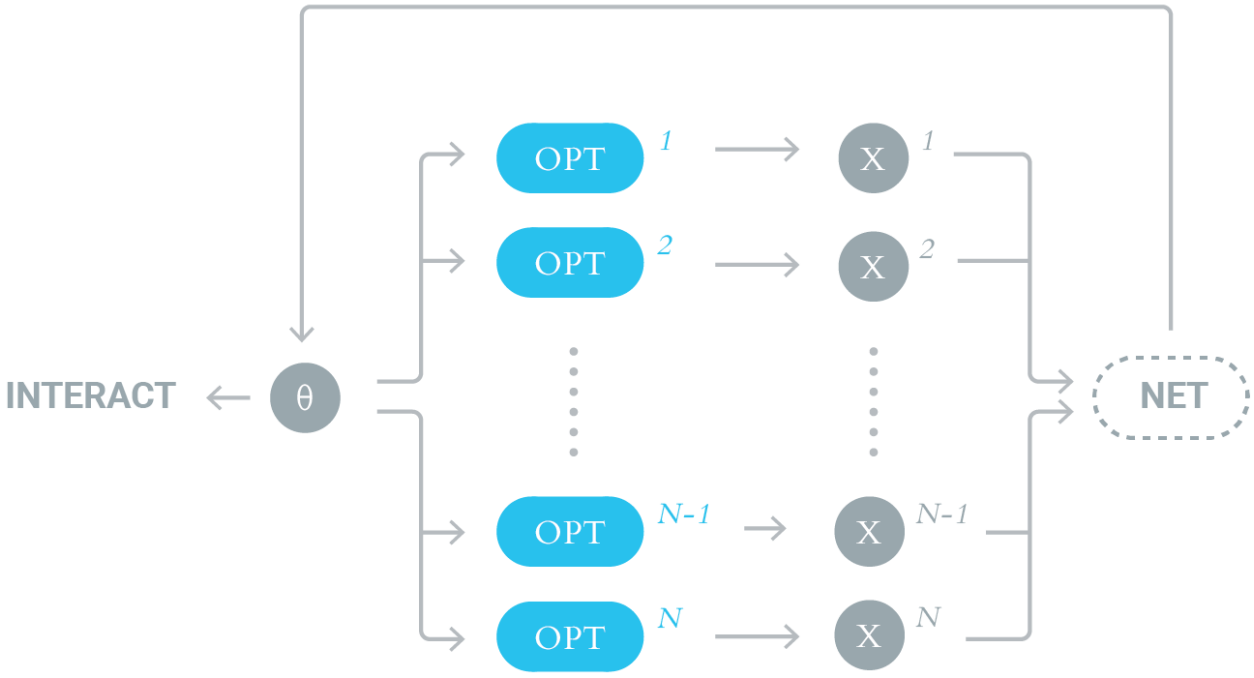


Figure 7.2: Overview of my approach. A neural network training machine (NET) generates network parameters θ representing the policy learned from a character's optimal trajectories $\mathbf{X}$. These parameters are used for generating new trajectories close to the policy (OPT), as well as for interactive character control.

Let the state of the character be defined as $[\mathbf{q}\ \mathbf{f}\ \mathbf{r}]$, where $\mathbf{q}$ is the physical pose of the character

(root position, orientation and joint angles), $\mathbf{f}$ are the contact forces being applied on the character by the ground, and $\mathbf{r}$ is the recurrent memory state of the character. The motion of the character is a state trajectory of length T defined by $\mathbf{X} = \begin{bmatrix}\mathbf{q}^0 \ \mathbf{f}^0 \ \mathbf{r}^0 \ ... \ \mathbf{q}^T \ \mathbf{f}^T \ \mathbf{r}^T\end{bmatrix}$. Let $\mathbf{X}^1, ..., \mathbf{X}^N$ be a collection of N trajectories, each starting with different initial conditions and executing a different task (such as moving the character to a particular location).

I introduce a neural network control policy $\boldsymbol{\pi}_\theta : \mathbf{s} \mapsto \mathbf{a}$ parametrized by neural network weights θ that maps a sensory state of the character $\mathbf{s}$ at each point in time to an optimal action $\mathbf{a}$ that controls the character. In general, the sensory state can be designed by the user to include arbitrary informative features, but in this preliminary work I use the following simple representation:

$$\mathbf{s}^t = \begin{bmatrix}\mathbf{q}^t \ \mathbf{r}^t \ \dot{\mathbf{q}}^{t-1} \ \mathbf{f}^{t-1}\end{bmatrix} \qquad\qquad \mathbf{a}^t = \begin{bmatrix}\dot{\mathbf{q}}^t \ \dot{\mathbf{r}}^t \ \mathbf{f}^t\end{bmatrix},$$

where, e.g., $\dot{\mathbf{q}}^t \triangleq \mathbf{q}^{t+1} - \mathbf{q}^t$ denotes the instantaneous rate of change of $\mathbf{q}$ at time t. With this representation of the action, the policy directly commands the desired velocity of the character and applied contact forces, and determines the evolution of the recurrent state $\mathbf{r}$. Thus, my network learns both optimal controls and a model of dynamics.

Let $C_i(\mathbf{X})$ be the total cost of the trajectory $\mathbf{X}$, which rewards accurate execution of task i and physical realism of the character's motion. I want to jointly find a collection of optimal trajectories that each complete a particular task, along with a policy $\boldsymbol{\pi}_\theta$ that is able to reconstruct the sense and action pairs $\mathbf{s}^t(\mathbf{X})$ and $\mathbf{a}^t(\mathbf{X})$ of all trajectories at all timesteps:

$$\begin{aligned} &\underset{\theta \ \mathbf{X}^1 \ ... \ \mathbf{X}^N}{\text{minimize}} \quad \sum_i C_i(\mathbf{X}^i) \\ &\text{subject to} \quad \forall \ i, t: \ \mathbf{a}^t(\mathbf{X}^i) = \boldsymbol{\pi}_\theta(\mathbf{s}^t(\mathbf{X}^i)). \end{aligned} \tag{7.1}$$

The optimized policy parameters θ can then be used to execute policy in real-time and interactively control the character by the user.

7.1.1 Stochastic Policy and Sensory Inputs

Injecting noise has been shown to produce more robust movement strategies in graphics and optimal control [49, 136], reduce overfitting and prevent feature co-adaptation in neural network train-

ing [43], and stabilize recurrent behaviour of neural networks [46,115]. I inject noise in a principled way to aid in learning policies that do not diverge when rolled out at execution time.

In particular, I inject additive Gaussian noise into the sensory inputs $\mathbf{s}$ given to the neural network. Let the sensory noise be denoted $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_{\varepsilon}^2 I)$, so the resulting noisy policy inputs are $\mathbf{s} + \boldsymbol{\varepsilon}$. This change in input also induces a change in the optimal action to take. If the noise is small enough, the optimal action at nearby noisy states is given by the first order expansion

$$\mathbf{a}(\mathbf{s} + \boldsymbol{\varepsilon}) = \mathbf{a} + \mathbf{a}_{\mathbf{s}}\boldsymbol{\varepsilon}, \tag{7.2}$$

where $\mathbf{a}_{\mathbf{s}}$ (alternatively $\frac{d\mathbf{a}}{d\mathbf{s}}$) is the matrix of optimal feedback gains around $\mathbf{s}$. These gains can be calculated as a byproduct of trajectory optimization as described in section 7.2.2. Intuitively, such feedback helps the neural network trainer to learn a policy that can automatically correct for small deviations from the optimal trajectory.

7.1.2 Distributed Stochastic Optimization

The resulting constrained optimization problem (7.1) is nonconvex and too large to solve directly. I replace the hard equality constraint with a quadratic penalty with weight α:

$$R(\mathbf{s}, \mathbf{a}, \theta, \boldsymbol{\varepsilon}) = \frac{\alpha}{2} \left\| (\mathbf{a} + \mathbf{a}_{\mathbf{s}}\boldsymbol{\varepsilon}) - \boldsymbol{\pi}_{\theta}(\mathbf{s} + \boldsymbol{\varepsilon}) \right\|^2, \tag{7.3}$$

leading to the relaxed, unconstrained objective

$$\underset{\theta\ \mathbf{X}^1\ \ldots\ \mathbf{X}^N}{\text{minimize}} \sum_i C_i(\mathbf{X}^i) + \sum_{i,t} R(\mathbf{s}^t(\mathbf{X}^i), \mathbf{a}^t(\mathbf{X}^i), \theta, \boldsymbol{\varepsilon}^{i,t}). \tag{7.4}$$

I then proceed to solve the problem in block-alternating optimization fashion, optimizing for one set of variables while holding others fixed. In particular, I independently optimize for each $\mathbf{X}^i$ (trajectory optimization) and for θ (neural network regression).

As the target action $\mathbf{a} + \mathbf{a}_{\mathbf{s}}\boldsymbol{\varepsilon}$ depends on the optimal feedback gains $\mathbf{a}_{\mathbf{s}}$, the noise $\boldsymbol{\varepsilon}$ is resampled after optimizing each policy training sub-problem. In principle the noisy sensory state and corresponding action could be recomputed within the neural network training procedure, but I found it expedient to freeze the noise during NN optimization (so that the optimal feedback gains need

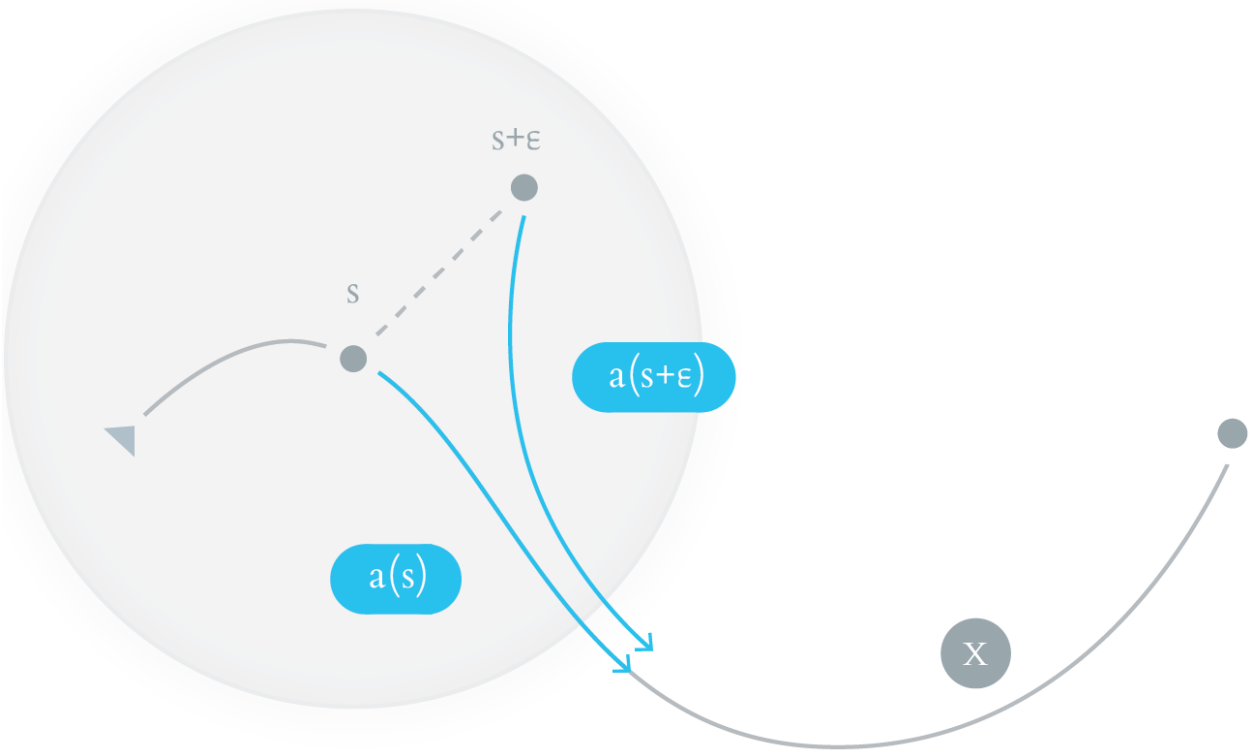


Figure 7.3: For a given optimal trajectory $\mathbf{X}$, the optimal actions $\mathbf{a}$ change in the neighborhood of $\mathbf{s}$ to compensate for deviation ε.

not be passed to the NN training process). Similar to recent stochastic optimization approaches in machine learning, I introduce quadratic proximal regularization terms (weighted by η) that keep the solution of the current iteration close to its previous optimal value. The resulting algorithm is

Algorithm 1 Distributed Stochastic Optimization

Sample sensor noise $\bar{\boldsymbol{\varepsilon}}^{i,t}$ for each t and i.

Solve N independent trajectory optimizations (section 7.2): $\bar{\mathbf{X}}^i = \operatorname{argmin}_{\mathbf{X}} C_i(\mathbf{X}) + \sum_t R(\mathbf{s}^{i,t}, \mathbf{a}^{i,t}, \bar{\boldsymbol{\theta}}, \bar{\boldsymbol{\varepsilon}}^{i,t}) + \frac{\eta}{2} \left\| \mathbf{X} - \bar{\mathbf{X}}^i \right\|^2$

Solve neural network regression (section 7.3): $\bar{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} \sum_{i,t} R(\bar{\mathbf{s}}^{i,t}, \bar{\mathbf{a}}^{i,t}, \boldsymbol{\theta}, \bar{\boldsymbol{\varepsilon}}^{i,t}) + \frac{\eta}{2} \left\| \boldsymbol{\theta} - \bar{\boldsymbol{\theta}} \right\|^2$

Repeat.

Thus I have reduced a complex policy search problem in (7.1) to an alternating sequence of

independent trajectory optimization and neural network regression problems, each of which are well-studied and allow the use of existing implementations.

7.2 Trajectory Optimization

I wish to find trajectories that start with particular initial conditions and execute the task, while satisfying physical realism of the character's motion. The existing approach I use is Contact-Invariant Optimization developed in the previous chapters, which is a direct trajectory optimization method based on inverse dynamics. Define the total cost for a trajectory $\mathbf{X}$:

$$C(\mathbf{X}) = \sum_t c(\boldsymbol{\phi}^t(\mathbf{X})), \tag{7.5}$$

where $\boldsymbol{\phi}^t(\mathbf{X})$ is a function that extracts a vector of features (such as root forces, contact distances, control torques, etc.) from the trajectory at time t and $c(\boldsymbol{\phi})$ is the state cost over these features.

Physical realism is achieved by satisfying equations of motion, non-penetration, and force complementarity conditions at every point in the trajectory [98]:

$$\begin{aligned} H(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) &= \boldsymbol{\tau} + J^\top(\mathbf{q}, \dot{\mathbf{q}})\mathbf{f} \\ \mathbf{d}(\mathbf{q}) &\geq \mathbf{0} \\ \mathbf{d}(\mathbf{q})^\top \mathbf{f} &= \mathbf{0} \\ \mathbf{f} &\in \mathbf{K}(\mathbf{q}), \end{aligned} \tag{7.6}$$

where $\mathbf{d}(\mathbf{q})$ is the distance of the contact to the ground and $\mathbf{K}$ is the contact friction cone. These constraints are implemented as soft constraints, as in the previous chapters and are included in $C(\mathbf{X})$. Initial conditions are also implemented as soft constraints in $C(X)$. Additionally I want to make sure the task is satisfied, such as moving to a particular location while minimizing effort. These task costs are the same for all my experiments and are described in section 7.7.

7.2.1 Optimal Trajectory

The trajectory optimization problem consists of finding the optimal trajectory parameters $\mathbf{X}$ that minimize the total cost (7.5):

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmin}}\, C(\mathbf{X}). \tag{7.7}$$

I solve the above optimization problem using Newton's method, which requires the gradient and Hessian of the total cost function. Using the chain rule, these quantities are

$$C_{\mathbf{X}} = \sum_t c^t_{\boldsymbol{\phi}} \boldsymbol{\phi}^t_{\mathbf{X}} \tag{7.8}$$

$$\begin{aligned} C_{\mathbf{XX}} &= \sum_t {\boldsymbol{\phi}^t_{\mathbf{X}}}^\top c^t_{\boldsymbol{\phi}\boldsymbol{\phi}} \boldsymbol{\phi}^t_{\mathbf{X}} + c^t_{\boldsymbol{\phi}} \boldsymbol{\phi}^t_{\mathbf{XX}} \\ &\approx \sum_t {\boldsymbol{\phi}^t_{\mathbf{X}}}^\top c^t_{\boldsymbol{\phi}\boldsymbol{\phi}} \boldsymbol{\phi}^t_{\mathbf{X}}, \end{aligned} \tag{7.9}$$

where the truncation of the last term in (7.9) is the common Gauss-Newton Hessian approximation [16]. I choose cost functions for which $c_{\boldsymbol{\phi}}$ and $c_{\boldsymbol{\phi}\boldsymbol{\phi}}$ can be calculated analytically. On the other hand, $\boldsymbol{\phi}_{\mathbf{X}}$ is calculated by finite differencing. The optimum can then be found by the following recursion:

$$\mathbf{X}^* = \mathbf{X}^* - C^{-1}_{\mathbf{XX}} C_{\mathbf{X}}. \tag{7.10}$$

Because this optimization is only a sub-problem (step 2 in algorithm 1), I don't run it to convergence, and instead take between one and ten iterations.

7.2.2 Optimal Feedback Gains

In addition to the optimal trajectory, I also need to find optimal feedback gains $\mathbf{a_s}$ necessary to generate optimal actions for noisy inputs in (7.2). While these feedback gains are a byproduct of indirect trajectory optimization methods such as LQG, they are not an obvious result of direct trajectory optimization methods like CIO. While I can use Linear Quadratic Gaussian (LQG) pass around my optimal solution to compute these gains, this is inefficient as it does not make use of computation already performed during direct trajectory optimization. Moreover, I found the resulting process can produce very large and ill-conditioned feedback gains. One could change the

objective function for the LQG pass when calculating feedback gains to make them smoother (for example, by adding explicit trajectory smoothness cost), but then the optimal actions would be using feedback gains from a different objective. Instead, I describe a perturbation method that reuses computation done during direct trajectory optimization, also producing better-conditioned gains. This is a general method for producing feedback gains that stabilize resulting optimal trajectories and can be useful for other applications.

Suppose I perturb a certain aspect of optimal trajectory $\mathbf{X}$, such that the sensory state changes: $\mathbf{s}(\mathbf{X}) = \bar{\mathbf{s}}$. I wish to find how the optimal action $\mathbf{a}(\mathbf{X})$ will change given this perturbation. I can enforce the perturbation with a soft constraint of weight λ, resulting in an augmented total cost:

$$\tilde{C}(\mathbf{X}, \bar{\mathbf{s}}) = C(\mathbf{X}) + \frac{\lambda}{2} \left\| \mathbf{s}(\mathbf{X}) - \bar{\mathbf{s}} \right\|^2 . \tag{7.11}$$

Let $\tilde{\mathbf{X}}(\bar{\mathbf{s}}) = \operatorname{argmin}^*_{\mathbf{X}} \tilde{C}(\mathbf{X}^*)$ be the optimum of the augmented total cost. For $\bar{\mathbf{s}}$ near $\mathbf{s}(\mathbf{X})$ (as is the case with local feedback control), the minimizer of augmented cost is the minimizer of a quadratic around optimal trajectory $\mathbf{X}$

$$\begin{aligned}
\tilde{\mathbf{X}}(\bar{\mathbf{s}}) &= \mathbf{X} - \tilde{C}^{-1}_{\mathbf{XX}}(\mathbf{X}, \bar{\mathbf{s}}) \tilde{C}_{\mathbf{X}}(\mathbf{X}, \bar{\mathbf{s}}) \\
&= \mathbf{X} - (C_{\mathbf{XX}} + \lambda \mathbf{s}_{\mathbf{X}}^\top \mathbf{s}_{\mathbf{X}})^{-1} (C_{\mathbf{X}} + \lambda \mathbf{s}_{\mathbf{X}}^\top (\mathbf{s}(\mathbf{X}) - \bar{\mathbf{s}})),
\end{aligned}$$

where all derivatives are calculated around $\mathbf{X}$. Differentiating the above w.r.t. $\bar{\mathbf{s}}$,

$$\begin{aligned}
\tilde{\mathbf{X}}_{\bar{\mathbf{s}}} &= \lambda (C_{\mathbf{XX}} + \lambda \mathbf{s}_{\mathbf{X}}^\top \mathbf{s}_{\mathbf{X}})^{-1} \mathbf{s}_{\mathbf{X}}^\top \\
&= C^{-1}_{\mathbf{XX}} \mathbf{s}_{\mathbf{X}}^\top (\mathbf{s}_{\mathbf{X}} C^{-1}_{\mathbf{XX}} \mathbf{s}_{\mathbf{X}}^\top + \frac{1}{\lambda} I)^{-1},
\end{aligned}$$

where the last line follows from Woodbury identity and has the benefit of reusing $C^{-1}_{\mathbf{XX}}$, which is already computed as part of trajectory optimization. The optimal feedback gains for $\mathbf{a}$ are $\mathbf{a}_{\bar{\mathbf{s}}} = \mathbf{a}_{\mathbf{X}} \tilde{\mathbf{X}}_{\bar{\mathbf{s}}}$. Note that $\mathbf{s}_{\mathbf{X}}$ and $\mathbf{a}_{\mathbf{X}}$ are subsets of $\boldsymbol{\phi}_{\mathbf{X}}$, and are already calculated as part of trajectory optimization. Thus, computing optimal feedback gains comes at very little additional cost.

Our approach produces softer feedback gains according to parameter λ without modifying the cost function. The intuition is that instead of holding initial state fixed (as LQG does, for example), I make matching the initial state a soft constraint. By weakening this constraint, I can modify initial state to better achieve the master cost function without using very aggressive feedback.

7.3 Neural Network Policy Regression

After performing trajectory optimization, I perform standard regression to fit a neural network to the noisy fixed input and output pairs $\{\mathbf{s}+\boldsymbol{\varepsilon},\ \mathbf{a}+\mathbf{a}_{\mathbf{s}}\boldsymbol{\varepsilon}\}^{i,t}$ for each timestep and trajectory. My neural network policy has a total of K layers, hidden layer activation function σ (tanh, in the present work) and hidden units $\mathbf{h}^k$ for layer k. The network's parameters are $\boldsymbol{\theta} = \left[\mathbf{W}^1\ \mathbf{b}^1\ ...\ \mathbf{W}^K\ \mathbf{b}^K\right]$, its input is $\mathbf{h}^0 = \mathbf{s}$, and its output is $\mathbf{h}^K = \boldsymbol{\pi}_\theta(\mathbf{s}) = \mathbf{a}$. To learn a model that is robust to small changes in neural state, I add independent Gaussian noise $\boldsymbol{\gamma}^k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_\gamma^2 I)$ with variance $\boldsymbol{\sigma}_\gamma^2$ to the neural activations at each layer during training. Wager et al. [132] observed this noise model makes hidden units tend toward saturated regions and less sensitive to precise values of individual units. This type of noise is also commonly used in neural network models from the computational neuroscience community [46, 115]. The network is evaluated with the forward recursion

$$\begin{aligned}
&\mathbf{h}^0 = \mathbf{s}, \\
&\mathbf{z}^k = \mathbf{W}^k\mathbf{h}^{k-1} + \mathbf{b}^k \qquad\qquad \mathbf{h}^k = \sigma(\mathbf{z}^k) + \boldsymbol{\gamma}^k \\
&\mathbf{h}^K = \mathbf{W}^K\mathbf{h}^{K-1} + \mathbf{b}^K.
\end{aligned}$$

It is trained to minimize the proximally regularized error

$$\sum_{i,t} R(\mathbf{s}^t(\mathbf{X}^i), \mathbf{a}^t(\mathbf{X}^i), \theta, \boldsymbol{\varepsilon}^{i,t}) + \frac{\eta}{2}\left\|\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}\right\|^2,$$

the gradient of which can be computed with backpropagation (taking into account the noise $\boldsymbol{\gamma}$).

Note that the policy is queried in the trajectory optimization process (equation (7.3 in line 2 of algorithm 1 depends on $\mathbf{X}$) and thus I need to compute the first derivatives (Jacobian) of $\boldsymbol{\pi}_\theta$ with respect to its input $\mathbf{A}^0 = \frac{\partial}{\partial \mathbf{h}^0}\boldsymbol{\pi}_\theta(\mathbf{h}^0)$. It can be calculated with the following backward recursion:

$$\begin{aligned}
&\mathbf{A}^K = I \\
&\mathbf{A}^k = \operatorname{diag}\left[\sigma'(\mathbf{z}^k)\right](\mathbf{W}^{k+1})^\top \mathbf{A}^{k+1} \\
&\mathbf{A}^0 = (\mathbf{W}^1)^\top \mathbf{A}^1.
\end{aligned}$$

The neural network weight matrices are initialized with a spectral radius of just above 1, similar to [46, 115, 116]. This helps to make sure initial network dynamics are stable and do not vanish or explode.

As with the trajectory optimization sub-problems, I do not run the neural network trainer to convergence but rather perform only a single pass of batched stochastic gradient descent over the dataset before updating the parameters $\boldsymbol{\theta}$ in step 3 of Algorithm 1.

All my experiments use 5 layer neural networks with 250 hidden units in each layer. This may be larger than necessary for some of my simpler experiments, but I have intentionally used such oversized network to remove the need to tune network size per experiment. Importantly, these oversized networks do not suffer from overfitting and still produce good policies that generalize because of the injected sense and activation noise when training.

7.4 Training Trajectory Generation

To train a neural network for interactive use, I required a data set that includes dynamically changing task's goal state. The task, in this case, is the locomotion of a character to a movable goal position controlled by the user. (Our character's goal position was always set to be the origin, which encodes the characters state position in the goal position's coordinate frame. Thus the "origin" may shift relative to the character, but this keeps behavior invariant to the global frame of reference.)

Our trajectory generation creates a dataset consisting of trials and segments. Each trial k starts with a reference physical pose and null recurrent memory $[\mathbf{q}\ \dot{\mathbf{q}}\ \mathbf{r}]^{\text{init}}$ and must reach goal location $\mathbf{g}^{k,0}$. After generating an optimal trajectory $\mathbf{X}^{k,0}$ according to section 7.2, a random timestep t is chosen to branch a new segment with $[\mathbf{q}\ \dot{\mathbf{q}}\ \mathbf{r}]^{t}$ used as the initial recurrent memory. A new goal location $\mathbf{g}^{k,1}$ is also chosen randomly for optimal trajectory $\mathbf{X}^{k,1}$.

This process represents the character changing direction at some point along its original trajectory plan: "interaction" in this case is simply a new change in goal position. This technique allows for my initial states and goals to come from the distribution that reflects the character's typical motion. In all my experiments, I use between 100 to 200 trials, each with 5 branched segments.

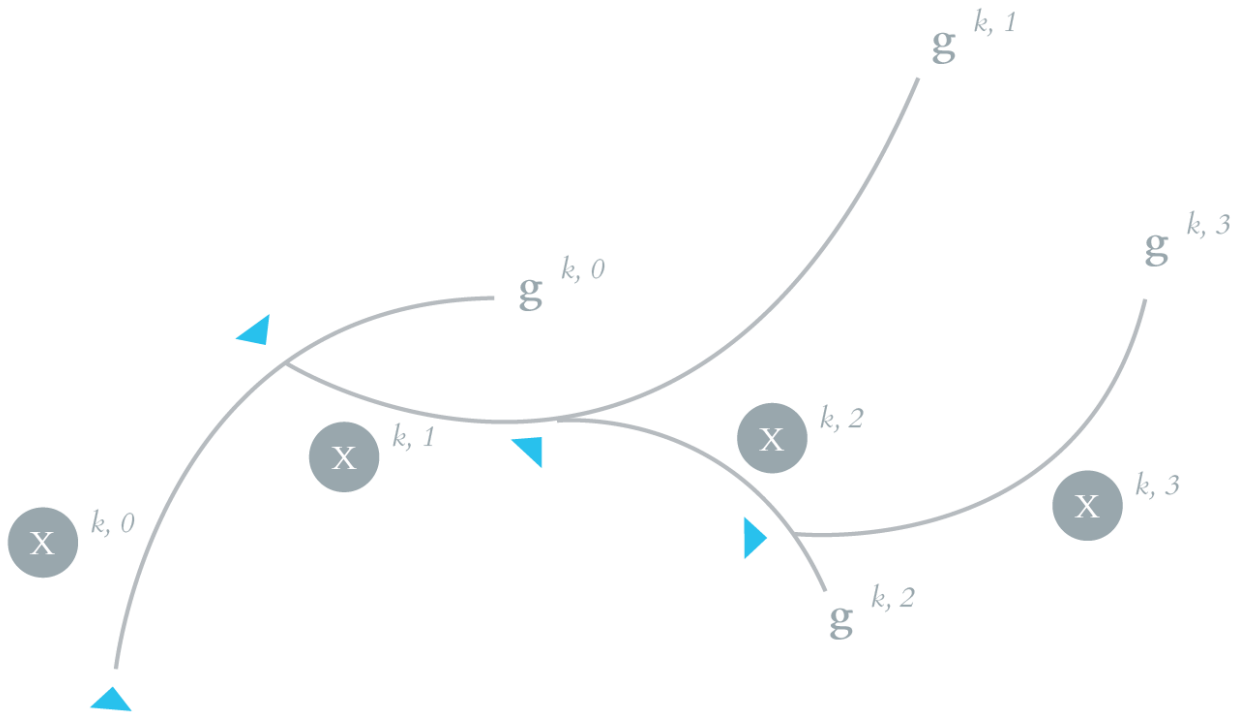


Figure 7.4: Generating rich behavior suitable for interaction training, my optimization nodes first find an optimal trajectory $\mathbf{X^{k,0}}$ from the initial state to goal $\mathbf{g}^{k,0}$. From this trajectory, it randomly initiates a new segment (indicated by the blue triangle) towards a new goal $\mathbf{g}^{k,1}$. This segment is optimized to become trajectory $\mathbf{X^{k,1}}$, and so on to generate as much data as required.

7.5 Distributed Training Architecture

Our training algorithm was implemented in a asynchronous, distributed architecture, utilizing a GPU for neural network training. Simple parallelism was achieved by distributing the trajectory optimization processes to multiple node machines, while the resulting data was used to train the NN policy on a single GPU node. (See figure 7.2.)

Amazon Web Service's EC2 3.8xlarge instances provided the nodes for optimization, while a g2.2xlarge instance provided the GPU. Utilizing a star-topology with the GPU instance at the center, a Network File System server distributes the training data $\mathbf{X}$ and network parameters θ to necessary processes within the cluster. Each optimization node is assigned a subset of the total trials and segments for the given task. This simple usage of files for data storage meant no

supporting infrastructure other than standard file locking for concurrency.

I used a custom GPU implementation of stochastic gradient descent (SGD) to train the neural network control policy. For the first training epoch, all trajectories and action sequences are loaded onto the GPU, randomly shuffling the order of the frames. Then the neural network parameters θ are updated using batched SGD in a single pass over the data to reduce the objective in (7.4). At the start of subsequent training epochs, trajectories which have been updated by one of the trajectory optimization processes (and injected with new sensor noise ε) are reloaded.

Although this architecture is asynchronous, the proximal regularization terms in the objective prevent the training data and policy results from changing too quickly and keep the optimization from diverging. As a result, I can increase my training performance linearly for the size of cluster I am using, to about 30 optimization nodes per GPU machine. I run the overall optimization process until the average of 200 trajectory optimization iterations has been reached across all machines. This usually results in about 11000 neural network training epochs, and takes about 2.5 hours to complete, depending on task parameters and number of nodes.

7.6 Policy Execution

Once I find the optimal policy parameters $\boldsymbol{\theta}$ offline, I can execute the resulting policy in real-time under user control. Unlike non-parametric methods like motion graphs or Gaussian Processes, I do not need to keep any trajectory data at execution time. Starting with an initial state $\mathbf{X}^0$, I compute sensory state $\mathbf{s}$ and query the policy (without noise) for the desired action $\left[\dot{\mathbf{q}}^{\text{des}}\ \dot{\mathbf{r}}^{\text{des}}\ \mathbf{f}\right]$.

To evolve the physical state of the system, I directly optimize the next state $\mathbf{X}^1$ to match $\dot{\mathbf{q}}^{\text{des}}$ while satisfying equations of motion

$$\mathbf{X}^1 = \underset{\mathbf{X}}{\operatorname{argmin}} \left\|\dot{\mathbf{q}} - \dot{\mathbf{q}}^{\text{des}}\right\| \quad \text{subject to (7.6)}$$

Note that this is simply the optimization problem (7.7) with horizon $T = 1$, which can be solved at real-time rates and does not require any additional implementation. This approach is reminiscent of feature-based control [2, 29, 79].

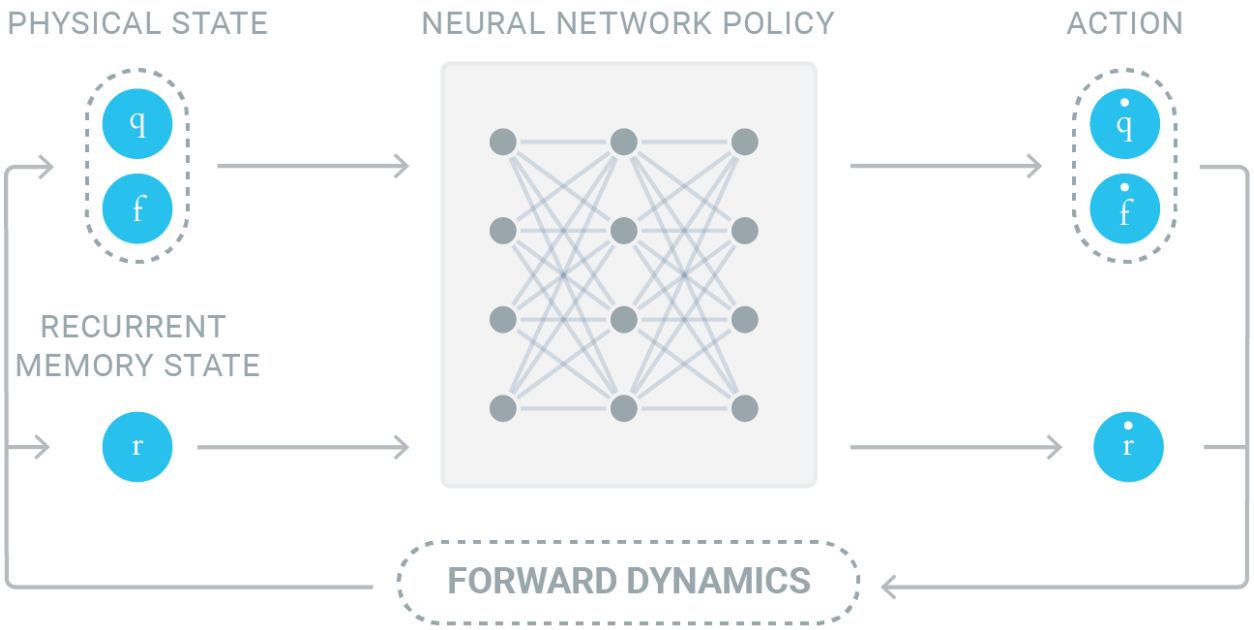


Figure 7.5: During real-time interactive use, the character's physical and recurrent memory state are evaluated in my neural network policy for a desired action. This action is evaluated with respect to forward dynamics to advance time.

The dynamics of recurrent memory evolve according to first-order dynamics directly controlled by action $\dot{\mathbf{r}}^{\text{des}}$:

$$\mathbf{r}^{t+1} = \mathbf{r}^{t} + \dot{\mathbf{r}}^{t,\text{des}} \Delta t.$$

Because my physical state evolution is a result of optimization (similar to an implicit integrator), it does not suffer from instabilities or divergence as Euler integration would, and allows the use of larger timesteps (I use Δt of 50ms in all my experiments). In the current work, the dynamics constraints are enforced softly and thus may include some root forces in simulation.

7.7 Results

This algorithm was applied to increasingly complex characters with locomotion tasks. The goal of each experiment was to learn a policy that allows for interactive locomotion. In terms of cost this means reducing goal position and orientation, minimizing torque usage, minimizing contact

σ_ε	1e-2
σ_γ	1e-2
α	1e1
λ	1e2
η	1e-2

Table 7.1: Algorithmic Constants

goal pose	1e-1
center of mass vel.	1e-3
torque	1e-4
contact force	1e-4

Table 7.2: Optimization Cost Weights

forces, and minimizing the center of mass velocity. Interaction meant that the goal pose relative to the character changes according to user input. The experiments started with a swimming turtle and flying bird creature before evolving into biped and quadruped walking tasks. I used the MuJoCo physics simulator [127] engine for my dynamics calculations. For reproducibility, the values of the algorithmic constants used in all experiments are given in table 7.7.

7.7.1 Swimming

Our swimming creature featured four fins with two degrees of freedom each, centered around a simple box body. This configuration is most akin to a sea turtle. It learned to navigate in a three dimensional volume towards a goal position and orientation. The creature is propelled by lift and drag forces for simulated water density of 1000kg/m^3. To maintain itself at a goal position, it would sweep down opposite fins in a cyclical patter, treading water.

7.7.2 Flying

Our bird creature was a modification of the swimmer, with opposing two-segment wings. The creature was similar to the swimmer, but the medium density is changed to that of air ($1.2kg/m^3$). Position goal was be maintained with a cyclical flapping motion (more vigorous now, because of the lower density). While state-machines or central pattern generators can utilize cyclic motions, my characters also learned to utilize lift forces to coast to distant goal positions, as well as modulate flapping speed to change altitude.

7.7.3 Biped Locomotion

Three bipedal creatures were created to explore the algorithm's function with respect to contact forces. Two creatures were akin to a humanoid - one large and one small, both with arms - while the other had a very wide torso compared to its height. All characters learned to walk to the target pose and would utilize a limit cycle gait. This alternating left/right footstep cycle behavior emerged without any user input or hand-crafting.

7.7.4 Quadruped Locomotion

This algorithm learned a stereotypical trot gait for a dog-like quadruped. For a spider-like quadruped, the gait maintained a typical opposite legged activation pattern. Again, these gaits emerged without user input and reflect what is generated by optimal trajectories for similar tasks.

7.7.5 Neural Network Training

To test the efficacy of my joint optimization technique, I compared this algorithm to naive neural network training on a static optimal trajectory dataset. I used the scene parameters of a biped character walking and disabled neural network training, generating trajectories as according to 7.4. Then, I performed my regression on this static data set with no new trajectories being optimized.

While the training error does decrease with the separately trained neural network, it plateaus much more quickly than my joint optimization. (See figure 7.6.) By enforcing my joint optimiza-

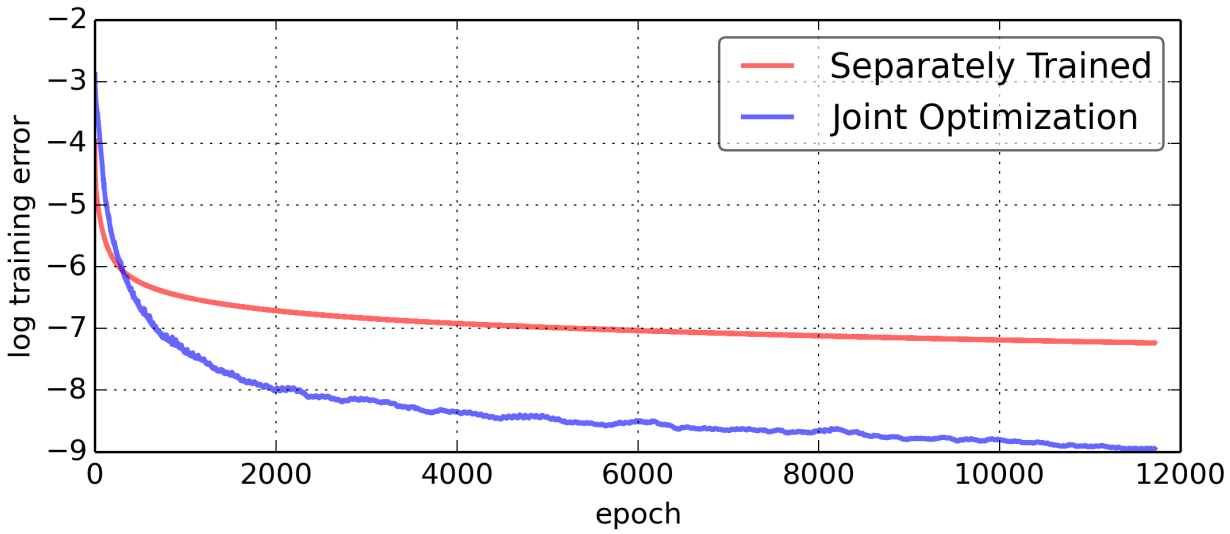


Figure 7.6: Instead of training a policy network and generating trajectories simultaneously, I first generated a large set of trajectories without a policy before training a neural network policy.

tion's trajectory generation to be close to my current neural network policy, I ensure that these trajectories are not inconsistent with what was previously trained on, while still completing the desired task. As a result, the joint optimization training error is significantly lower.

The result of separately trained neural networks is the likelihood of a policy returning action states that diverge from the optimal trajectories, as seen in figure 7.7. Joint optimization's policy returns actions near optimal throughout the trajectory; separate training results in actions that often diverge quickly to extreme results.

7.7.6 Effect of state noise and feedback

I wish to further compare policies trained without state noise, and those trained with state noise and action feedback. In figure 7.8 one can see typical ball trajectories generated by rolling out policies trained without state noise and those trained with state noise, respectively. I find that even though the first type of network matches the target controls very well, the derivatives of network output wrt input $\frac{\partial \pi}{\partial \mathbf{s}}$ behave very irregularly (red plot in 7.9). On the other hand, feedback gains $\bar{U}_X$

(dashed green in figure 7.9)are a much more regular signal that my method matches in addition to controls.

7.7.7 Comparison to Model-Predictive Control

Model-predictive control (MPC) allows another real-time controller for task-driven character behavior. In fact, the trajectory costs for both MPC and my method are very similar. The resulting trajectories, however, end up being different: MPC creates effective trajectories that are not cyclical. This suggests a significant nullspace of task solutions, but from all these solutions, my joint optimization - especially the cost terms of matching the neural network output - act to regularize trajectory optimization to regularly sensible actions.

7.8 Discussion and Future Work

I have presented an automatic way of generating neural network parameters that represent a control policy for physically consistent interactive character control, only requiring a dynamical character model and task description. Using both trajectory optimization and stochastic neural networks together combines correct behavior with real-time interactive use. Furthermore, the same algorithm and controller architecture can provide interactive control for multiple creature morphologies.

While the behavior of the characters reflected efficient task completion in this work, additional modifications could be made to affect the style of behavior – costs during trajectory optimization can affect how a task is completed. Incorporation of muscle actuation effects into my character models may result in more biomechanically plausible actions for that (biologically based) character. In addition to changing the character's physical characteristics, I could explore different neural network architectures and how they compare to biological systems. With this work, I have networks that enable diverse physical action, which could be augmented to further reflect biological sensorimotor systems [56]. This model could be used to experiment with the effects of sensor delays and the resulting motions, for example [37].

The work in this chapter focused on locomotion of different creatures with the same algo-

rithm. Previous chapter has demonstrated behaviors such as getting up, climbing, and reaching with contact-invariant optimization. Real-time policies using this chapter's algorithm could allow interactive use of these behaviors as well. Extending beyond character animation, this work could be used to develop controllers for robotics applications that are robust to sensor noise and perturbations if the trained character model accurately reflects the robot's physical parameters.

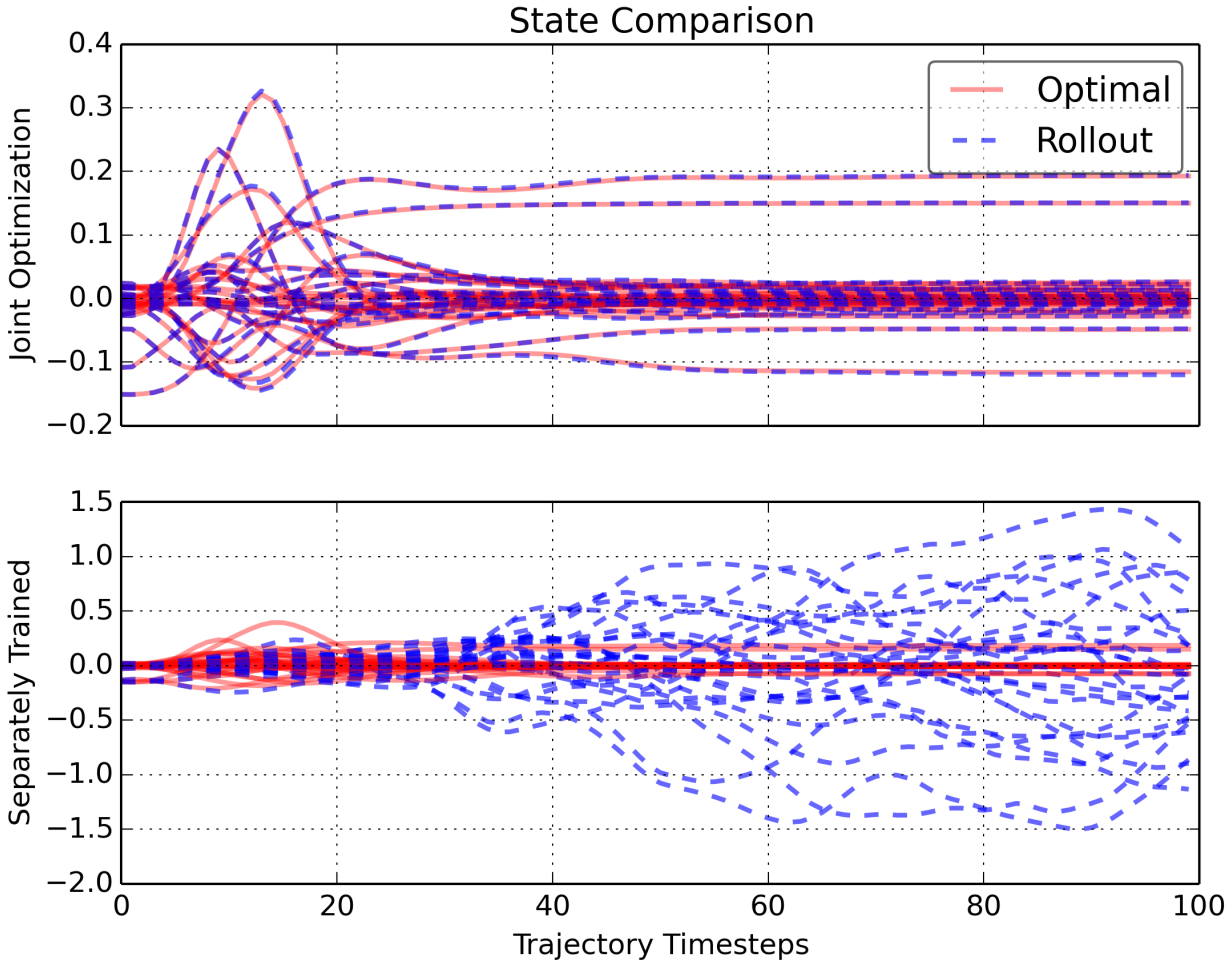


Figure 7.7: As a result, the deviation of the policy rollout from the optimized trajectory is significant, and may grow through the action.

Figure 7.8: Ball trajectories generated by training without (left) and with (right) injecting state noise and feedback

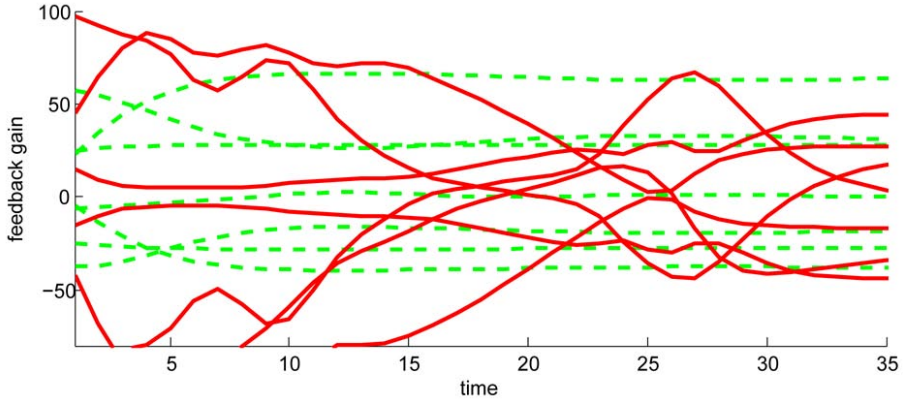


Figure 7.9: Optimal feedback gains for a typical ball trajectory (dashed green) and the corresponding $\frac{\partial \pi}{\partial \mathbf{s}}$ produced by a standard neural network trained without injecting state noise (red).

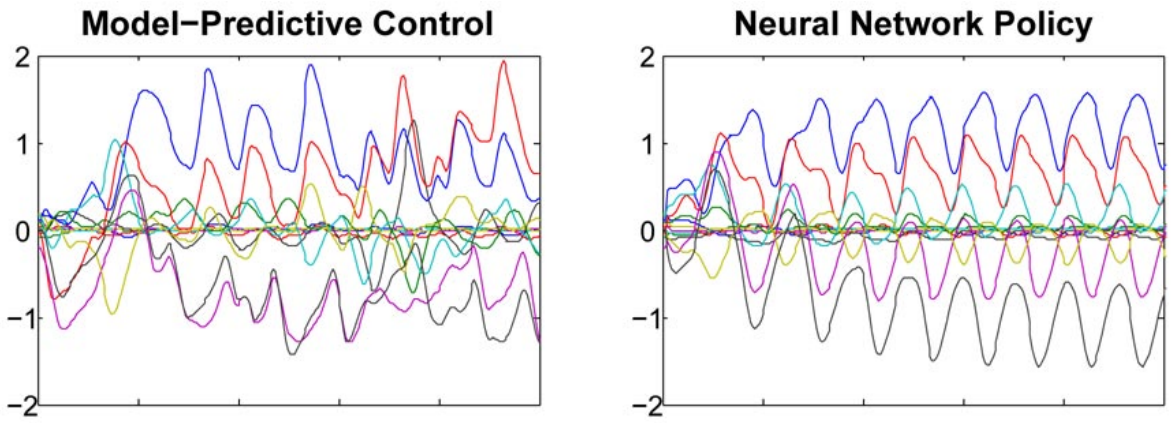


Figure 7.10: I compare the velocities that result from MPC and my method. While I can assume that both trajectories successfully complete the given task, my method generates trajectories that are cyclic and predictable.

Chapter 8
CONCLUSIONS

I presented a fully automated framework for synthesizing a wide range of movement behaviors and learning neural network parameters that represent a control policy for physically consistent interactive character control, only requiring a dynamical character model and task description. The framework is agnostic to the morphology of the character, and indeed I showed that movement behaviors can be created for significantly different types of characters using the same algorithm and controller architecture. Using both trajectory optimization and stochastic neural networks together combines correct behavior with real-time interactive use.

First key innovation and advantage of my framework is the simultaneous optimization of contacts and smooth portions of the movement. This was made possible by either introducing an auxiliary decision variable or a contact force variable for each potential contact. As a result, it was possible to optimize very long and temporally complex movement sequences that have previously remained beyond the reach of numerical optimization methods.

Second key innovation is a new method for combining the benefits of trajectory optimization and global policy (neural network) learning. The trajectory optimizer was given an augmented cost, making it find solutions that resemble the current output of the neural network - which in turn makes training the network easier. This introduces a trade-off that allows for degradation of task performance in order to make the learning problem easier. One exciting application of this to biological movement control is to investigate how human or animal motion is influenced by considerations of mental effort and capacity (whereas most previous work primarily focused on influence of metabolic energy expenditure). Other exciting avenues of future work include incorporation of other complex high-dimensional sensory modalities, such as vision and touch to the movement policies and inspecting how their influence behavior.

BIBLIOGRAPHY

[1] P. Abbeel, A. Coates, M. Quigley, and A. Ng. An application of reinforcement learning to aerobatic helicopter flight. *NIPS*, 2006.

[2] Yeuhi Abe, Marco da Silva, and Jovan Popović. Multiobjective control with frictional contacts. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 249–258, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[3] Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. Trajectory optimization for full-body movements with complex contacts. *IEEE Trans. Vis. Comput. Graph.*, 19(8), 2013.

[4] Robert McNeill Alexander. *Principles of Animal Locomotion*. Princeton University Press, 2003.

[5] K. N. An, K. Takahashi, T. P. Harrigan, and E. Y. Chao. Determination of muscle orientations and moment arms. *Journal of Biomechanical Engineering*, 106(3):280–282, 1984.

[6] Frank C. Anderson and Marcus G. Pandy. A dynamic optimization solution for vertical jumping in three dimensions. *Computer Methods in Biomechanics and Biomedical Engineering*, 2(3):201–231, 1999.

[7] Frank C. Anderson and Marcus G. Pandy. Dynamic optimization of human walking. *Journal of Biomechanical Engineering*, 123(5):381–390, 2001.

[8] Frank Clayton Anderson. *A dynamic optimization solution for a complete cycle of normal gait.* PhD thesis, University of Texas at Austin, 1999.

[9] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1997.

[10] Lindsay J. Bhargava, Marcus G. Pandy, and Frank C. Anderson. A phenomenological model for estimating metabolic energy consumption in muscle contraction. *Journal of Biomechanics*, 37:81–88, 2004.

[11] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. *International Conference on Robotics and Automation*, pages 348–353, 2000.

[12] M. Brubaker, L. Sigal, and D. Fleet. Estimating contact dynamics. *International Conference on Computer Vision*, 2009.

[13] K. Byl and R. Tedrake. Metastable walking machines. *International Journal Robotics Research*, 2009.

[14] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5), 1995.

[15] G. A. Cavagna, P. A. Willems, and N. C. Heglund. Walking on Mars. *Nature*, 393(6686), 1998.

[16] Pei Chen. Hessian matrix vs. gauss-newton hessian matrix. *SIAM J. Numerical Analysis*, 49(4):1417–1435, 2011.

[17] Joel E. Chestnutt, Philipp Michel, James J. Kuffner, and Takeo Kanade. Locomotion among dynamic obstacles for the honda ASIMO. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 29 - November 2, 2007, Sheraton Hotel and Marina, San Diego, California, USA*, pages 2572–2573, 2007.

[18] Matei T. Ciocarlie and Peter K. Allen. Hand posture subspaces for dexterous robotic grasping. *International Journal Robotics Research*, 28(7):851–867, 2009.

[19] Michael F. Cohen. Interactive spacetime control for animation. In *Computer Graphics*, volume 26, pages 293–302. ACM, 1992.

[20] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive dynamic walkers. *Science*, 2005.

[21] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. *ACM Transactions on Graphics*, 28(5), 2009.

[22] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Transactions on Graphics*, 29(4), 2010.

[23] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4), 2011.

[24] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel Van De Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics (TOG)*, 30(4):59, 2011.

[25] Mark R. Cutkosky and Robert D. Howe. Dextrous robot hands. chapter Human grasp choice and robotic grasp analysis, pages 5–31. Springer-Verlag New York, 1990.

[26] Marco da Silva, Frédo Durand, and Jovan Popović. Linear bellman combination for control of character animation. *ACM Transactions on Graphics*, 28(3), 2009.

[27] Saltiel P. dAvella, A. and E. Bizzi. Combinations of muscle synergies in the construction of a natural motor behavior. *Nature Neuroscience*, 6:300–308, 2003.

[28] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Transactions on Graphics*, 29(4), 2010.

[29] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Transactions on Graphics (TOG)*, 29(4):131, 2010.

[30] Scott L. Delp, Peter Loan, Melissa G. Hoy, Felix E. Zajac, Eric L. Topp, and Joseph M. Rosen. An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures. *IEEE Transactions on Biomedical Engineering*, 37(8):757–767, 1990.

[31] L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. *ICASSP*, 2013.

[32] Rodger J. Elble, Charles Moody, Keith Leffler, and Raj Sinha. The initiation of normal walking. *Movement Disorders*, 9(2), 1994.

[33] Tom Erez and Emanuel Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 4914–4919. IEEE/RSJ, 2012.

[34] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proc. SIGGRAPH*. ACM, 2001.

[35] Anthony C. Fang and Nancy Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics*, 22(3):417–426, 2003.

[36] Hartmut Geyer and Hugh Herr. A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(3):263–273, 2010.

[37] Hartmut Geyer and Hugh Herr. A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 18(3):263–273, 2010.

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

[39] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 522–531, New York, NY, USA, 2004. ACM.

[40] Christopher M. Harris and Daniel M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394:780–784, 1998.

[41] Kris K. Hauser, Timothy Bretl, Jean-Claude Latombe, Kensuke Harada, and Brian Wilcox. Motion planning for legged robots on varied terrain. *I. J. Robotic Res.*, 27(11-12):1325–1349, 2008.

[42] Chris Hecker, Bernd Raabe, Ryan W Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. In *ACM Transactions on Graphics (TOG)*, volume 27, page 27. ACM, 2008.

[43] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[44] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *Proc. SIGGRAPH*, 1997.

[45] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proc. SIGGRAPH*. ACM, 1995.

[46] Gregor M. Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. *Cerebral Cortex*, 2012.

[47] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. *Proc AIAA*, 2007.

[48] GE Hovland, S Hanssen, E Gallestey, S Moberg, T Brogardh, S Gunnarsson, and M Isaksson. Nonlinear identification of backlash in robot transmissions. In *Proceedings of the 33rd ISR (International Symposium on Robotics)*, 2002.

[49] Dongsung Huh and E. Todorov. Real-time motor control using recurrent neural networks. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL '09. IEEE Symposium on*, pages 42–49, March 2009.

[50] A. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 2008.

[51] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review, 2008.

[52] Sumit Jain and C. Karen Liu. Controlling physics-based characters using soft contacts. *ACM Transactions on Graphics*, 30(6), 2011.

[53] R. Johansson and J. Flanagan. Coding and use of tactile signals from the fingertips in object manipulation tasks. *Nature Reviews Neuroscience*, 10:345359, 2009.

[54] Eunjung Ju, Jungdam Won, Jehee Lee, Byungkuk Choi, Junyong Noh, and Min Gyu Choi. Data-driven control of flapping flight. *ACM Trans. Graph.*, 32(5):151:1–151:12, October 2013.

[55] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2001: Expanding the Societal Role of Robotics in the the Next Millennium, Maui, HI, USA, October 29 - November 3, 2001*, pages 239–246, 2001.

[56] Eric Richard Kandel, James Harris Schwartz, Thomas M. Jessell, and Sarah Mack, editors. *Principles of neural science*. McGraw-Hill Medical, New York, Chicago, San Francisco, 2013.

[57] Z. Kolter, A. Coates, A. Ng, Y. Gu, and C. DuHadway. Space-indexed dynamic programming: Learning to follow trajectories. *ICML*, 2008.

[58] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt. Capturability-based analysis and control of legged locomotion. part 1: Theory and application to three simple gait models. *International Journal Robotics Research*, 2012.

[59] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM transactions on graphics (TOG)*, 21(3):473–482, 2002.

[60] Lucas Kovar, John Schreiner, and Michael Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, pages 97–104, New York, NY, USA, 2002. ACM.

[61] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[62] James J. Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Motion planning for humanoid robots. In *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*, pages 365–374, 2003.

[63] Arthur D. Kuo. A simple model of bipedal walking predicts the preferred speed-step length relationship. *Journal of Biomechanical Engineering*, 123(3):264–269, 2001.

[64] J. Kutch and F. Valero-Cuevas. Challenges and new approaches to proving the existence of muscle synergies of neural origin. *PLoS Computational Biology*, 8(5), 2012.

[65] M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 2003.

[66] Andrea N. Lay, Chris J. Hass, and Robert J. Gregor. The effects of sloped surfaces on locomotion: a kinematic and kinetic analysis. *Journal of Biomechanics*, 39, 2006.

[67] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Transactions on Graphics*, 28(4), 2009.

[68] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. Motion fields for interactive character locomotion. In *ACM SIGGRAPH Asia 2010 Papers*, SIGGRAPH ASIA '10, pages 138:1–138:8, New York, NY, USA, 2010. ACM.

[69] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *ICML '14: Proceedings of the 31st International Conference on Machine Learning*, 2014.

[70] Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics*, 31(4):28, 2012.

[71] C. Karen Liu. Synthesis of interactive hand manipulation. *Symposium on Computer Animation*, pages 163–171, 2008.

[72] C. Karen Liu. Dextrous manipulation from a grasping pose. *ACM Trans. Graph.*, 28(3), 2009.

[73] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics*, 24(3), 2005.

[74] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics*, 21(3):408–416, 2002.

[75] D. Liu and E. Todorov. Evidence for the flexible sensorimotor strategies predicted by optimal feedback control. *Journal of Neuroscience*, 27:9354–9368, 2007.

[76] K. Liu, A. Hertzmann, and Z. Popovic. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics*, 24:1071–1081, 2005.

[77] D. Lockhart and L. Ting. Optimal feedback transformations for balance. *Nature Neuroscience*, 10:1329–1336, 2007.

[78] Venkadesan M and Valero-Cuevas F. Neural control of motion-to-force transitions with the fingertip. *Journal of Neuroscience*, 28:1366–1373, 2008.

[79] Adriano Macchietto, Victor Zordan, and Christian R. Shelton. Momentum control for balance. *ACM Trans. Graph.*, 28(3):80:1–80:8, July 2009.

[80] I. Manchester, U. Mettin, F. Iida, and R. Tedrake. Stable dynamic walking over uneven terrain. *International Journa of Robotics Research*, 30:265–279, 2011.

[81] R. Margaria and G. A. Cavagna. Human locomotion in subgravity. *Aerospace Medicine*, 35, 1964.

[82] Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 29(3):245–262, September 2003.

[83] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *Internatiional Journal of Robotics Research*, 2012.

[84] Matthew Millard and Scott L. Delp. A computationally efficient muscle model. In *ASME Summer Bioeng. Conf.*, 2012.

[85] Matthew Millard, Thomas Uchida, Ajay Seth, and Scott L. Delp. Flexing computational muscle: Modeling and simulation of musculotendon dynamics. *Journal of Biomech. Eng.*, 135, 2013.

[86] Andrew T. Miller and Peter K. Allen. Examples of 3d grasp quality computations. *International Conference on Robotics and Automation*, pages 1240–1246, 1999.

[87] Andrew T. Miller and Andrew T. Miller. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11:110–122, 2004.

[88] W. Miller, R. Sutton, and P. Werbos. *Neural Networks for Control*. MIT Press, 1990.

[89] Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics*, 29(3), 2010.

[90] U. Muico, Y. Lee, J. Popovic, and Z. Popovic. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.*, 28(3), 2009.

[91] Uldarico Muico, Jovan Popović, and Zoran Popović. Composite control of physically simulated characters. *ACM Transactions on Graphics*, 30(3), 2011.

[92] L. Bianchi N. Borghese and F. Lacquaniti. Kinematic determinants of human locomotion. *Journal of Physiology*, 494:863–879, 1996.

[93] Allison M. Okamura, Niels Smaby, and Mark R. Cutkosky. An overview of dexterous manipulation. *International Conference on Robotics and Automation*, pages 255–262, 2000.

[94] Jacquelin Perry and Judith Burnfield. *Gait Analysis: Normal and Pathological Function*. Slack Incorporated, 2010.

[95] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

[96] Nancy S. Pollard and Victor B. Zordan. Physically based grasping control from example. *Symposium on Computer Animation*, pages 311–318, 2005.

[97] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proceedings of SIGGRAPH 99, Annual Conference Series*, pages 11–20. ACM, 1999.

[98] Michael Posa and Russ Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, pages 527–542. Springer, 2013.

[99] J. Pratt, J. Carff, S. Drakunov, and A. Goswami. Capture point: A step toward humanoid push recovery. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 200–207, Dec 2006.

[100] J. Rathelot and P. Strick. Subdivisions of primary motor cortex based on cortico-motoneuronal cells. *Proceedings of the National Academy of Sciences*, 106(3):918–923, 2009.

[101] John R Rebula, Peter D Neuhaus, Brian V Bonnlander, Matthew J Johnson, and Jerry E Pratt. A controller for the littledog quadruped walking on rough terrain. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1467–1473. IEEE, 2007.

[102] Charles F. Rose, Peter-Pike J. Sloan, and Michael F. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. *Comput. Graph. Forum*, 20(3):239–250, 2001.

[103] Stphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011.

[104] Alla Safonova, Jessica K. Hodgins, and Nancy Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics*, 23(3):514–521, 2004.

[105] Marco Santello, Martha Fl, and John F. Soechting. F.: Postural hand synergies for tool use. *The Journal of Neuroscience*, 18(3):10105–10115, 1998.

[106] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Transactions of the Royal Society B*, 2003.

[107] Justin E. Seipel and Philip Holmes. Running in three dimensions: Analysis of a point-mass sprung-leg model. *I. J. Robotic Res.*, 24(8):657–674, 2005.

[108] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint*, 2014.

[109] J. Si, A. Barto, W. Powell, and D. Wunsch. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, 2004.

[110] Manoj Srinivasan and Andy Ruina. Computer optimization of a minimal biped model discovers walking and running. *Nature*, 2005.

[111] Manoj Srinivasan and Andy Ruina. Computer optimization of a minimal biped model discovers walking and running. *Nature*, 439:72–75, 2006.

[112] B. Stepehns and C. Atkeson. Push recovery by stepping for humanoid robots with force controlled joints. *Humanoids*, 2010.

[113] Shinjiro Sueda, Andrew Kaufman, and Dinesh K. Pai. Musculotendon simulation for hand animation. *SIGGRAPH*, pages 83:1–83:8, 2008.

[114] Adnan Sulejmanpašić and Jovan Popović. Adaptation of performed ballistic motion. *ACM Trans. Graph.*, 24(1), 2005.

[115] David Sussillo and L.F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544 – 557, 2009.

[116] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.

[117] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000.

[118] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA, 1998.

[119] Jie Tan, Yuting Gu, C. Karen Liu, and Greg Turk. Learning bicycle stunts. *ACM Trans. Graph.*, 33(4):50:1–50:12, 2014.

[120] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *IROS*, 2012.

[121] Y. Tassa and E. Todorov. Stochastic complementarity for local control of discontinuous dynamics. In *Proc. RSS*, 2010.

[122] E. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, pages 3137–3181, 2010.

[123] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: a physics engine for model-based control. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[124] E. Todorov and Z. Ghahramani. Analysis of the synergies underlying complex hand manipulation. *IEEE Engineering in Medicine and Biology*, pages 4637–4640, 2004.

[125] Emanuel Todorov. A convex, smooth and invertible contact model for trajectory optimization. *International Conference on Robotics and Automation*, pages 1071–1076, 2011.

[126] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE/RSJ, 2012.

[127] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS'12*, pages 5026–5033, 2012.

[128] M. Tresch and A. Jarc. The case for and against muscle synergies. *Curr Opin Neurobiol*, 19:601–607, 2009.

[129] M. Tresch, P. Saltiel, and E. Bizzi. The construction of movement by the spinal cord. *Nature Neuroscience*, 2:162–167, 1999.

[130] Michiel van de Panne and Alexis Lamouret. Guided optimization for balanced locomotion, 1995.

[131] M. Vukobratovic and B. Borovac. Zero-moment point thirty-five years of its life. *International Journal of Human Robotics*, 2004.

[132] S. Wager, S. Wang, and P. Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[133] Kevin Wampler, Jovan Popović, and Zoran Popović. Animal locomotion controllers from scratch. In *Computer Graphics Forum*, volume 32, pages 153–162. Wiley Online Library, 2013.

[134] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics*, 28(3), 2009.

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

[136] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29(4):73:1–73:8, July 2010.

[137] Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics*, 31(4), 2012.

[138] Michael Vande Weghe, Matthew Rogers, Michael Weissert, and Yoky Matsuoka. *International Conference on Robotics and Automation*, pages 3375–3379, 2004.

[139] A. Witkin and M. Kass. Spacetime constraints. *SIGGRAPH*, 1988.

[140] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics*, volume 22, pages 159–168. ACM, 1988.

[141] W. Wooten. *Simulation of Leaping, Tumbling, Landing, and Balancing Humans*. PhD thesis, Georgia Institute of Technology, 1998.

[142] Jia-chi Wu and Zoran Popović. Realistic modeling of bird flight animations. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 888–895. ACM, 2003.

[143] Jia-chi Wu and Zoran Popović. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics*, 29(4), 2010.

[144] K. Yamane, J. Kuffner, and J. Hodgins. Synthesizing animations of human manipulation tasks, journal = SIGGRAPH, year = 2004.

[145] Yuting Ye and C. Karen Liu. Optimal feedback control for character animation using an abstract model. *ACM Transactions on Graphics*, 29(4), 2010.

[146] Yuting Ye and Karen Liu. Synthesis of detailed hand manipulations using contact sampling. *SIGGRAPH*, 2012.

[147] K. Yin, K. Loken, and M. van de Panne. Simbicon: simple biped locomotion control. *ACM Transactions on Graphics*, page 105, 2007.

[148] KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Continuation methods for adapting simulated skills. *ACM Transactions on Graphics*, 27(3), 2008.

[149] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.