Composable Controllers for Physics-Based Character Animation

by

Petros Faloutsos

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Composable Controllers for Physics-Based Character Animation

Petros Faloutsos
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2002

An ambitious goal in the area of physics-based computer animation is the creation of virtual actors that autonomously synthesize realistic human motions and possess a broad repertoire of lifelike motor skills. To this end, the control of dynamic, anthropomorphic figures subject to gravity and contact forces remains a difficult open problem. We propose a framework for composing controllers in order to enhance the motor abilities of such figures. A key contribution of our composition framework is an explicit model of the "pre-conditions" under which motor controllers are expected to function properly. We demonstrate controller composition with pre-conditions determined not only manually, but also automatically based on Support Vector Machine (SVM) learning theory. We evaluate our composition framework using a family of controllers capable of synthesizing basic actions such as balance, protective stepping when balance is disturbed, protective arm reactions when falling, and multiple ways of standing up after a fall. We furthermore demonstrate these basic controllers working in conjunction with more dynamic motor skills within a two-dimensional and a three-dimensional prototype virtual stuntperson. Our composition framework promises to enable the community of physics-based animation practitioners to more easily exchange motor controllers and integrate them into dynamic characters.

Dedication

To my father, Nikolaos Faloutsos, my mother, Sofia Faloutsou, and my wife, Florine Tseu.

Acknowledgements

I am done! Phew! It feels great. I have to do one more thing and that is to write the acknowledgements, one of the most important parts of a PhD thesis. The educational process of working towards a PhD degree teaches you, among other things, how important the interaction and contributions of the other people are to your career and personal development. First, I would like to thank my supervisors, Michiel van de Panne and Demetri Terzopoulos, for everything they did for me. And it was a lot. You have been the perfect supervisors. THANK YOU! However, I will never forgive Michiel for beating me at a stair-climbing race during a charity event that required running up the CN Tower stairs. Michiel, you may have forgotten, but I haven't!

I am grateful to my external appraiser, Jessica Hodgins, and the members of my supervisory committee, Ken Jackson, Alejo Hausner and James Stewart, for their contribution to the successful completion of my degree.

I would like to thank my close collaborator, Victor Ng-Thow-Hing, for being the richest source of knowledge on graphics research, graphics technology, investing and martial arts movies. Too bad you do not like Jackie Chan, Victor.

A great THANKS is due to Joe Laszlo, the heart and soul of our lab's community spirit. Joe practically ran our lab during some difficult times. He has spent hours of his time to ensure the smooth operation of the lab and its equipment. I am also grateful to Joe for tons of inspiring discussions, and for performing all kinds of stunts that I needed to see for my thesis work. His performance has been ...forever captured in this thesis.

I would also like to thank all the DGP lab members for creating an amazing research environment. Thanks Eugene Fiume, Michael Neff, Glenn Tsang, Meng Sun, Chris Trendal, David Mould, Corina Wang, Ryan Meredith-Jones, Anastasia Bezerianos, Paolo Pacheco, monica schraefel, Alejo Hausner, Sageev Oore, David Modjeska. Glenn, thanks for being the BZFlag darklord and for getting upset when I called you a "scavenger", I loved it.

A lot of thanks is due to the Greek gang's past and present members: Periklis Andritsos, Theodoulos Garefalakis, Panayiotis Tsaparas, Vassilis Tzerpos, Spyros Angelopoulos, Stergios Anastasiadis, Angeliki Maglara, Rozalia Christodoulopoulou, Anastasia Bezerianos, Georgos Katsirelos, Georgos Chalkiadakis, Georgos Giakoupis, Giannis Papoutsakis, Giannis Velegrakis, Tasos Kementsietsidis, Fanis Tsandilas, Mixalis Flouris, Nora Jantschukeite, Themis Palpanas, Giannis Lazaridis, Giannis Kassios, Anna Eulogimenou, Verena Kantere, and whoever I am forgetting. Theo, thanks for laughing with my jokes. Panayioti, thanks for proving that time travel is possible if you are late enough. Vassili, thanks for the giouvarlakia that you cooked long time ago. Perikli, thanks for cooking pastitsio. Themi, thanks for not cooking. Lazaridaki, one day I WILL touch your basketball. Special thanks to my office mates, Rozalia and Periklis, for putting up with my gym bag.

Thanks to my old friends in Greece, Penny Anesti, Gwgw Liassa, Yiannis Tsakos, Athanasios Stefos and Aleksandros Xatzigiannis. Yianni, it is time to tell you that that day in kindergarten I was not crying. I, strategically, pretended.

Our ex-graduate administrator, Kathy Yen, has made the early parts of my student life so much easier. Kathy, thank you very much for everything. Thanks also to our current graduate administrator, Linda Chow, for all her help.

Finally, I would like to thank my family, my partner in life Florine Tseu, brother emeritus Piotrek Gozdyra, Michalis Faloutsos, Christos Faloutsos, Christina Cowan, Maria Faloutsou, Antonis Mikrovas, Christos Mikrovas, Aleksis Kalamaras and most of all my father, Nikolaos Faloutsos, and my mother, Sofia Faloutsou. Guys, you have made this possible. Christo, thanks for all the advice! Michalis and Piotrek, thanks for everything! I would also like to thank Katherine Tseu, Irene Tseu, Dureen Tseu, for everything they have done for me. Thanks also to my mutts, Missa and Petra, for guarding our house from ferocious squirrels and for not eating my PhD thesis.

Contents

1 Introduction **1**
- 1.1 Autonomous Characters 1
- 1.2 Problem Statement 2
- 1.3 Methodology 2
- 1.4 Summary of Results 4
  - 1.4.1 Our virtual characters 4
  - 1.4.2 Control 6
- 1.5 Applications 10
- 1.6 Contributions 11
- 1.7 Thesis structure 11

2 Previous Work **12**
- 2.1 Biomechanics 12
- 2.2 Robotics 14
- 2.3 Computer Animation 15
- 2.4 Controller composition 17
- 2.5 Simulated control systems 18
  - 2.5.1 Commercial animation software 18
  - 2.5.2 Robotics 18

3 Composition Framework **20**
- 3.1 Composing controllers 20
- 3.2 Our composition framework 22
- 3.3 Controller abstraction 23
- 3.4 Pre-conditions 24
- 3.5 Post-conditions 24
- 3.6 Expected performance 25
- 3.7 Arbitration 25
- 3.8 Transitions 26
- 3.9 Determining pre-conditions 26
- 3.10 Manual approach 27
- 3.11 Discussion 28

4 Learning pre-conditions **29**
- 4.1 Machine learning 29
- 4.2 Learning the pre-conditions as a machine learning problem 30
- 4.3 Choosing a classification method 31

4.4 Support Vector Machines 31
4.5 Applying SVMs 33
4.6 Results 33

5 Simulation **38**
5.1 Physics-based simulation of articulated figures 38
5.1.1 Numerical solution of the equations of motion 39
5.2 Control 40
5.3 Design methods 40
5.4 Our control structures 41
5.5 Supervisor controller 42
5.5.1 Sensors 43
5.5.2 Command interface 44
5.6 Implementing the composable API 44
5.7 Default Controller 46
5.8 Everyday Actions 46
5.8.1 Balancing 47
5.8.2 Falling 48
5.8.3 Stand-to-sit and sit-to-crouch 49
5.8.4 Rising from a supine position 50
5.8.5 Rolling over 51
5.8.6 Rising from a prone position 52
5.8.7 Kneel-to-crouch 53
5.8.8 Step 54
5.8.9 Protective step 55
5.8.10 Crouch-to-stand 57
5.8.11 Double-stance-to-crouch 58
5.8.12 Walk 58
5.9 Stunts 59
5.9.1 The *kip* move 59
5.9.2 Plunging and rolling 61
5.10 Discussion 63

6 Dynamic Animation and Control Environment **65**
6.1 Motivation 65
6.2 Features 65
6.3 Component abstraction 67
6.3.1 Systems 67
6.3.2 Simulators 70
6.3.3 Actuators 73
6.3.4 Ground actuator 75
6.3.5 Musculotendon model 75
6.3.6 Geometries 76
6.4 Implementation 76
6.5 Who is DANCE for? 77

7 Results **79**
7.1 Robot sequence 79
7.2 Skeleton sequence 81
7.3 Multiple characters 82
7.4 Discussion 83

8 Conclusions and Future Work **84**
8.1 Planning 84
8.2 Multiple controllers 85
8.3 Training set 85
8.4 Expected performance and pre-conditions. 86
8.5 Additional testing 86
8.6 Future: Intelligent agents 87

A SVM^{light} parameters **88**

B SD/Fast description file for our 3D character **90**

C SD/Fast description file for our 2D character **93**

D DANCE script for the tackle example **96**

Bibliography **98**

List of Figures

1.1 Layers of an intelligent virtual character. 1
1.2 An overview of the system. 3
1.3 A dynamic "virtual stuntman" falls to the ground, rolls over, and rises to an erect position, balancing in gravity. 3
1.4 Dynamic models and their degrees of freedom (DOFs). 5
1.5 Controllers for the 2D character. 8
1.6 Controllers for the 3D character. 9

3.1 An abstract visualization of potential transitions between controllers for walking and running. 21
3.2 Degrees of continuity. 21
3.3 Motion curve blending. 21
3.4 Two level composition scheme. 22
3.5 Controller selection and arbitration during simulation. 24
3.6 Controllers and typical transitions for 3D figure 27
3.7 Controllers and typical transitions for 2D figure 27

4.1 Training set and actual boundary for a 2D problem. 30
4.2 Two dimensional SVM classifier. 32

5.1 An articulated character. 39
5.2 Controlling an articulated character. 40
5.3 A stand-sit-stand pose controller. 42
5.4 A few sensors associated with the 3D model. 43
5.5 Manual and learned approximations of the success region. 45
5.6 Critically damped balance controller. 47
5.7 Falling in different directions 48
5.8 Sitting and getting up from a chair. 54
5.9 Rising from a supine position on the ground and balancing erect in gravity. . . . 55
5.10 Taking a step. 55
5.11 The *kip* move performed by both a real and a virtual human. 60
5.12 Ouch! 62
5.13 Plunge and roll on a different terrain. 62
5.14 Different crouching configurations. 64

6.1 The architecture of DANCE. 67
6.2 Articulated figures in DANCE. 68
6.3 Working with articulated figures in DANCE. 69
6.4 Dynamic free-form deformations in DANCE. 70

6.5 A two-link ...saltshaker. 71
6.6 Working with articulated figures in DANCE. 74
6.7 A complex muscle actuator, courtesy of Victor Ng-Thow-Hing. 75
6.8 Class hierarchy in DANCE. 77

7.1 The terminator sequence, left to right and top to bottom. 80
7.2 A dynamic "virtual stuntman" falls to the ground, rolls over, and rises to an erect position, balancing in gravity. 82
7.3 Two interacting virtual characters. 82
7.4 Articulated and flexible characters. 83

8.1 A sequence of controllers chosen by a planner. 85

Chapter 1

Introduction

1.1 Autonomous Characters

An ambitious goal that is shared between a number of different scientific areas is the creation of virtual characters that autonomously synthesize realistic human-like motions and possess a broad repertoire of lifelike motor skills. Computer graphics, robotics, and biomechanics researchers are all interested in developing skillful human characters that can simulate a real human in terms of visual appearance, motor skills and ultimately reasoning and intelligence. Developing a simulated character of human complexity is an enormous task. Humans are capable of performing a very wide variety of motor control tasks ranging from picking up a small object, to complex athletic maneuvres such as serving a tennis ball, and many others, Schmidt and Wrisberg [94]. Determining appropriate motor control strategies so that a simulated character can reproduce them is surprisingly difficult even for everyday motions such as walking.

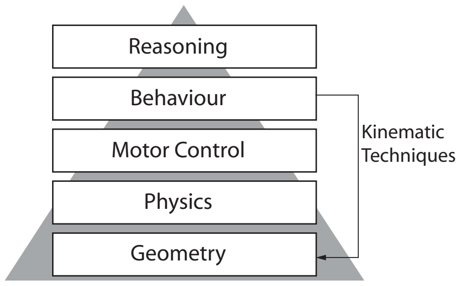


Figure 1.1: Layers of an intelligent virtual character.

A simulated human character can be described in terms of a number of hierarchical layers as presented in Figure 1.1. At the bottom of the hierarchy is the most tangible layer that simply models the character's visual appearance using suitable geometric representations. The *Physics* layer models the dynamic and physical properties of the character such as, muscle and body structure. It also provides the relation between the kinematic state of the character and the applied forces. The *Motor Control* layer is responsible for the coordination of the character's body so that it can perform desired motor tasks. The *Behavior* layer implements behaviors that are based on external stimuli and the character's intentions and internal state. Examples of such behaviors are "Eat when you are hungry" and "Move away from danger." The top layer, *Reasoning*, models the ability of a character to develop autonomously new behaviors based on reasoning and inference. It is worth noting that the classic kinematic animation techniques such as the ones used by the film industry, directly connect the behavior and the geometry layers, Figure 1.1. Characters are animated in a fashion similar to puppeteering or using precomputed motions. In contrast, the foundation of physics-based and robotic techniques is the motor control layer, which attempts to model the way real creatures coordinate their muscles in order to move about.

It is clear that the goal of developing skillful simulated characters is highly interdisciplinary and broad. The next section describes the specific problem we are trying to solve and where our work lies within the above hierarchy.

1.2 Problem Statement

The work in this thesis involves the motor control layer shown in Figure 1.1. Our goal is to work towards the development of virtual characters that have a wide portfolio of motor control skills. At the same time, we aim to implement an animation toolbox that allows practioners to build upon and re-use existing research results.

Developing complex, skillful, simulated characters is an enormous task. Complex characters such as humans, are capable of performing a great range [1] of sophisticated motions that can be very dynamic and highly optimized. Clearly, a divide-and-conquer technique is a promising way to tackle the problem. Developing robust parameterized motor control for such characters has been an active area of research both in robotics and in computer animation. However, the results are still limited. In addition, the isolation and separation of research results limits the progress in the area. Typically, research groups use their own custom software and characters and it is therefore difficult to share and reuse results. Because of the difficulty of producing good results there is a clear need for cooperation in the area. To realize a useful level of cooperation we need (a) a conceptual framework that allows the integration of multiple motion controllers in one functional set; and (b) a software system that can serve as the common platform for research in the area. These problems are the focus of this work.

1.3 Methodology

Our methodology is based on the idea of incrementally expanding the character's skills. We start with the basic motor skills, such as balancing upright, and work our way outwards towards more complex motions. Our framework allows researchers to implement controllers using their own techniques and add them in our system, thus realizing an incremental scheme where the character's repertoire of skills expands with every added controller.

We propose a simple framework for composing specialist controllers into more general and capable control systems for dynamic characters. In our framework, *individual controllers* are black boxes encapsulating control knowledge that is possibly gleaned from the biomechanics literature, derived from the robotics control literature, or developed specifically for animation control. Individual controllers must be able to determine two things: (1) a controller should be able to determine whether or not it can take the dynamic character from its current state to some desired goal state, and (2) an active controller should be able to determine whether it is operating nominally, whether it has succeeded, or whether it has failed. Any controller that can answer these queries may be added to a pool of controllers managed by a *supervisor*

[1]Schmidt and Wrisberg [94] provide a categorization of abilities that humans have in various degrees in order to perform everyday or athletic motions.

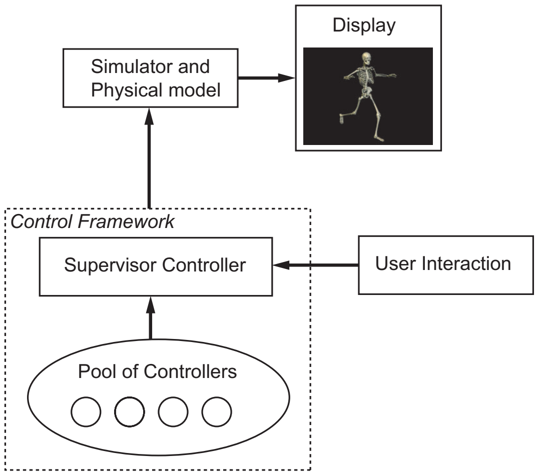


Figure 1.2: An overview of the system.

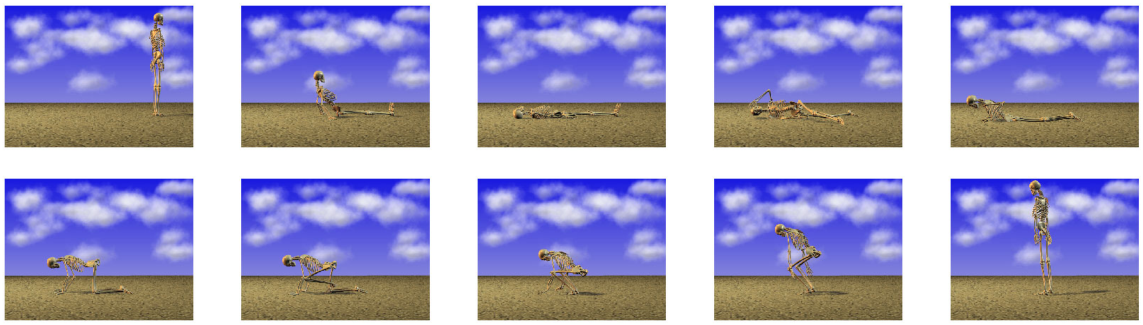


Figure 1.3: A dynamic "virtual stuntman" falls to the ground, rolls over, and rises to an erect position, balancing in gravity.

controller whose goal is to resolve more complex control tasks. The supervisor controller does not need to know specific information about each available controller, which allows controllers to be added in or removed from or at run time. Figure 1.2 shows a schematic representation of our system.

An important technical contribution within our controller composition framework is an explicit model of *pre-conditions*. Pre-conditions characterize those regions of the dynamic figure's state space within which an individual controller is able to successfully carry out its mission. Initially, we demonstrate the successful composition of controllers based on manually determined pre-conditions. We then proceed to investigate the question of whether pre-conditions can be determined automatically. We devise a promising solution which employs Support Vector Machine (SVM) learning theory. Our novel application of this technique learns appropriate pre-conditions through the repeated sampling of individual controller behavior in operation.

As a test bed for our techniques, we are developing a physically simulated animated character capable of a large repertoire of motor skills. An obvious application of such a character is the creation of a *virtual stuntperson*: the dynamic nature of typical stunts makes them dangerous to perform, but also makes them an attractive candidate for the use of physics-based animation. The open challenge here lies in developing appropriate control strategies for specific actions and ways of integrating them into a coherent whole.

We demonstrate families of composable controllers for articulated skeletons whose physical parameters reflect anthropometric data consistent with a fully fleshed adult male. One family of controllers is for a 37 degree-of-freedom (DOF) 3D articulated skeleton, while a second family of controllers has been developed for a similar 16 DOF 2D articulated skeleton. While the 3D skeleton illustrates the ultimate promise of the technique, the 2D skeleton is easier to control and thus allows for more rapid prototyping of larger families of controllers and more careful analysis of their operation. As evidenced by the number of past and present papers on controlling 2D walking and jumping, in the robotics literature, the control of broad skilled repertoires of motion remains very much an open problem even for 2D articulated figures. Having fewer degrees of freedom saves significant amounts of time, both during simulation and during the learning of the pre-conditions. In general, control and machine learning techniques for complex simulated human characters and high dimensional spaces are faced with the well-known *curse of dimensionality* [20], therefore using a simplified version of a problem improves greatly the efficiency of the algorithms used. The composition framework we are proposing makes no assumptions of dimension and has no knowledge of how the participating controllers work. Therefore it can handle both the two dimensional and the three dimensional case.

Figure 1.3 illustrates the 3D dynamic character autonomously performing a complex control sequence composed of individual controllers responsible for falling reactions, rolling-over, getting up, and balancing in gravity. The upright balancing dynamic figure is pushed backwards by an external force; its arms react protectively to cushion the impact with the ground; the figure comes to rest in a supine position; it rolls over to a prone position, pushes itself up on all fours, and rises to its feet; finally it balances upright once again. A subsequent disturbance will elicit similar though by no means identical autonomous behavior, because the initial conditions and external forces will usually not be exactly the same. Control sequences of such intricacy for fully dynamic articulated figures are unprecedented in the physics-based animation literature.

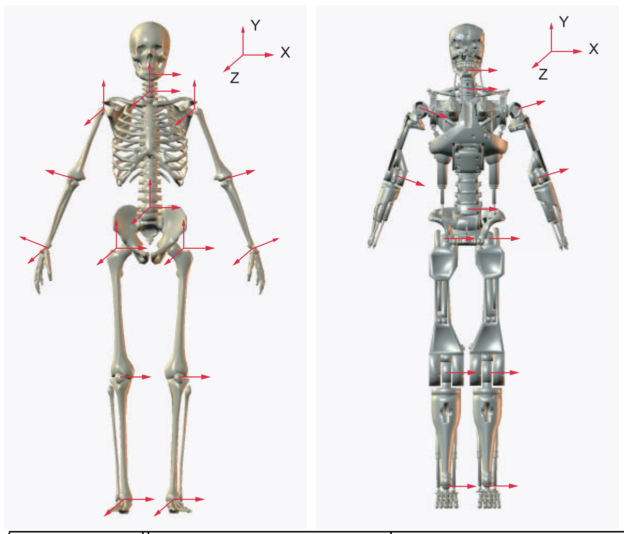
Our framework is built on top of DANCE, a software system that we have developed jointly with Victor Ng-Thow-Hing. We provide both the composition module and the base software system for free for non-commercial use with the hope that it will become the main tool for research in the area. In that case, practitioners will be able to share, exchange and integrate controllers. We believe that this can significantly advance the state of the art in physics-based character animation which is currently hampered by the segmentation and isolation of research results.

1.4 Summary of Results

This section provides an overview of our experiments and the results we are able to achieve with our method. We first describe our dynamic virtual characters and then their control.

1.4.1 Our virtual characters

Fig. 1.4 depicts our 2D and 3D articulated character models. The red arrows indicate the joint positions and axes of rotational degrees of freedom (DOFs) which are also enumerated in the table. The 3D skeleton model has 37 DOFs, six of which correspond to the global translation and rotation parameters. The table in Fig. 1.4 lists the DOFs for the skeleton and a 2D "terminator" model. The dynamic properties of both models, such as mass and moments of inertia, are taken from the biomechanics literature (see Winter [109]) and correspond to a fully-fleshed adult male. In particular, the total weight of each model is 89.57 kilograms. The movement of the rotational degrees of freedom of the models is restricted by the physical



Joint	*Rotational DOFs 3D skeleton model*	*Rotational DOFs 2D terminator model*
Head	1	1
Neck	3	1
Shoulder	2	1
Elbow	2	1
Wrist	2	-
Waist	3	1
Hip	3	1
Knee	1	1
Ankle	2	1

Figure 1.4: Dynamic models and their degrees of freedom (DOFs).

Joint	Axis	Lower limit	Upper limit
Waist	x	-45	90
Waist	z	-55	55
Waist	y	-50	50
Neck	x	-50	90
Neck	z	-60	60
Neck	y	-80	80
Head	x	-45	45
Right shoulder	z	-90	90
Right shoulder	y	-80	160
Right elbow	y	0	120
Right elbow	x	-90	40
Right hand	z	-90	90
Right hand	y	-45	45
Right thigh	x	-165	45
Right thigh	y	-120	20
Right thigh	z	-20	20
Right knee	x	0	165
Right foot	x	-45	50
Right foot	z	-2	35

Table 1.1: The joint limits of the 3D model.

limits of the human body. After using our own intuition and researching the literature we have decided to use the joint limits indicated in Table 1.1.

The equations of motion for both models are generated by SD/Fast [49] software, as described in Section 6.3.2. The script files used by the SD/Fast simulator compiler are given in Appendix B and Appendix C. They include all the details pertaining to our two dynamics models such as the mass, moments of inertia and dimensions of each body part.

1.4.2 Control

To test our framework we have implemented a number of composable controllers for a two dimensional and a three dimensional dynamic human model. The controllers for both models implement everyday motions such as taking steps and interesting stunts. The protective and falling behaviors that our simulated characters can perform when pushed along any arbitrary direction are of particular interest. The following controllers have been developed for the two dimensional character:

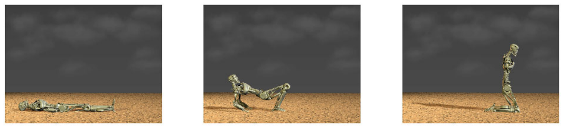
1. *Balance.* Maintains an upright stance using an inverted pendulum mode.
2. *Walk.* Takes a user-specified number of slow, deliberate steps.
3. *Dive.* Dives forward at a specified takeoff angle.
4. *ProtectStep.* When unbalanced it tries to take a step to maintain an upright stance.
5. *Fall.* Uses the arms to cushion falls in both forward and backward directions.
6. *ProneToKneel.* Takes the character from a prone position to a kneeling position.

7. *SupinetoKneel.* Takes the character from a supine position to a kneeling position.
8. *KneelToCrouch.* Takes the character from a kneeling position to a crouch.
9. *DoubleStanceToCrouch.* When the character is standing with one foot in front of the other it brings both feet side by side.
10. *CrouchToStand.* When the character is crouching, i.e. standing with its legs together but not straightened, this controller will straightened up the character and bring him to an upright stance.
11. *Sit.* Move from a standing position to a sitting position.
12. *SitToCrouch.* Move from a sitting position to a crouch.
13. *DefaultController.* Attempts to keep the character in a comfortable default position when no other controller can operate.

For the three dimensional character we have developed the following controllers:

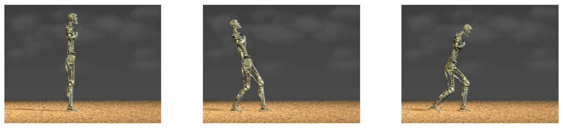
1. *Balance.* Maintains an up right stance using an inverted pendulum model.
2. *Step.* Takes one step forward for a specific starting state.
3. *Dive.* Dives forward at a specified takeoff angle.
4. *ProtectStep.* When unbalanced it tries to take a step to maintain an upright stance. It usually fails to maintain balance, but helps reduce the impact of the fall.
5. *Fall.* When falling it tries to absorb the shock using the arms. It is capable of handling falls in any direction.
6. *SupineToCrouch.* Takes the character from a supine [2] position to a crouch.
7. *ProneToCrouch.* Takes the character from a face down prone [3] position to a crouch.
8. *RollOver.* Takes the character from a supine to a face down prone position.
9. *CrouchToStand.* When the character is standing with its legs together but not straightened this controller straights the character.
10. *Kip.* A technique employed by gymnasts and martial artists to return to a standing position from a supine position.
11. *Sit.* Move from a standing position to a seated position.
12. *SitToCrouch.* Rise from a sitting position to a crouch.
13. *StandToAllFour.* Takes the character from a standing position to a position on the hands and knees.
14. *DefaultController.* Attempts to keep the character in a comfortable default position when no other controller can operate.

(a) Slow steps


(b) Supine to kneel.

(c) Prone to kneel.

(d) Kneel to crouch.


(e) Taking a protective step when pushed backwards.

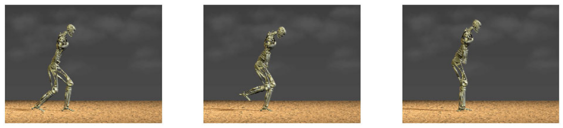

(f) Wide stance to crouch.

Figure 1.5: Controllers for the 2D character.

(a) Stand in place

(b) Stand to all fours.

(c) Prone to crouch.

(d) Crouch to stand.

(e) Roll over.

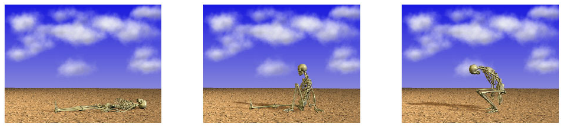
(f) Suicidal dive.

(g) Protective steps in various directions.

(h) Sit down.

(i) Sit to stand.

(j) Fall in various directions.


(k) Supine to crouch.

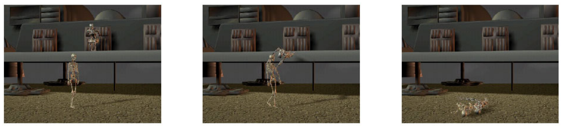

(l) Two interacting 3D characters.

Figure 1.6: Controllers for the 3D character.

Figures 1.5 and 1.6 show characteristic snapshots of the motions produced for 2D and the 3D models respectively.

1.5 Applications

Physics-based autonomous characters that can be directed to perform interesting motor tasks are an alternative to kinematically based animation methods. The physics-based aspect of these characters allows easier and more accurate modeling of the physical contact and collisions. Employing physical simulation avoids the repeatable and predictable motions that kinematic methods tend to produce. In addition, the realism and automation associated with physics-based techniques, can be very convenient for ballistic motions that involve collisions. In particular, the motions produced by our falls controllers when the character falls under gravity are physically accurate and completely automated.

Autonomous simulated agents that can be instructed to perform difficult or dangerous stunts are probably one of the best immediate applications for the film industry. A prime example of such motions is the dive-down-the-stairs sequence depicted in Figure 1.6 (f). Virtual actors have been used already in feature films. The movie *Titanic* by Paramount Studios and Twentieth Century Fox is one of the first movies to employ kinematically-driven digital actors to depict real people in a crowd scene.

Autonomous physics-based agents capable of performing sequences of complex motor tasks are also very important for the next generation of computer games. Parameterized kinematic motions are the most common method for animating interactive agents in the gaming industry. However, they are limited to precomputed variations of a nominal motion and they cannot model well physical contact and interaction. Sophisticated games that involve complex group behaviors, athletic skills and physical contact can potentially benefit from the use of physics-based skillful agents.

Biomechanics and robotics research have an ongoing interest in understanding and modeling the human body and motor control skills. Our control framework can also be potentially the basis for simulation and modeling in these areas, allowing researchers to combine and build upon existing results.

Education in simulation, control and animation techniques can significantly benefit from inexpensive and versatile tools. The DANCE software system is an open source tool that can be used easily in a classroom. It is freely available, portable, extensible and modular, allowing students to experiment with a variety of interesting problems. It can be used as the common platform that allows students to use existing results, share their results, collaborate on projects and visualize their work. DANCE's modular and plugin architecture renders it particularly suitable for collaborative projects and fun classroom competitions.

DANCE and its plugins have been used for research in graphics, biomechanics and robotics and for applications such as human simulation, virtual puppeteering, digitized muscle data visualization, implementation of inverse kinematics techniques for human animation and flexible object simulation. We believe that it can become the common platform for research in these areas facilitating collaboration and the re-use of existing results.

[2] At a *supine* position the character lies on his/her back facing up.

[3] At the *prone* position the character lies on his/her belly facing down.

1.6 Contributions

The main contributions of this thesis are summarized as follows:

- We propose and implement a framework for controller composition based on the concepts of *pre-conditions*, *post-conditions*, and *expected performance.*
- We implement a software system that implements this framework, and we offer it to the research community. It includes modules that perform collision detection[4] and resolution for arbitrary polygonal objects.
- We investigate the use of *support vector machines* for the classification of multi-dimensional state spaces for animation purposes.
- We implement physics-based controllers for reactive falling behaviors, interesting stunts, and everyday motions for a 2D and a 3D dynamic model.
- We demonstrate the successful use of our framework in composing these multiple controllers together to allow for 2D and 3D human models to exhibit integrated skills.

A minor contribution of this thesis is that we provide the research community with a freely available software for simulating a 36-DOF dynamic human model [5].

1.7 Thesis structure

The remainder of this thesis is organized as follows. After reviewing related prior work in Chapter 2, we present the details of our control framework in Chapter 3. We then investigate the question of determining pre-conditions in Chapter 4. Chapter 5 describes the controllers we have implemented and Chapter 6 presents our software system. Chapters 7 presents the details of the example in Figure 1.3 along with several other examples that demonstrate the effectiveness of our framework. Chapter 8 concludes this thesis and discusses avenues for future research opened up by our work.

[4]Based on the RAPID collision detection library.

[5]Symbolic Dynamics Inc. has agreed to the free distribution of the equations of motion of the model.

Chapter 2

Previous Work

The simulation and animation of human characters is a challenging problem in many respects. Comprehensive solutions must aspire to distill and integrate knowledge from biomechanics, robotics, control, and animation. Models for human motion must also meet a particularly high standard, given our familiarity with what the results should look like. Not surprisingly, a divide-and-conquer strategy is evident in most approaches, focusing efforts on reproducing particular motions in order to yield a tractable problem and to allow for comparative analysis.

Biomechanics, robotics and animation research share a number of common goals and problems with respect to understanding and modeling the human motor control system. They approach these problems from different angles. Biomechanics research focuses on medical accuracy and detail, robotics focuses on building skillful machines and animation focuses on developing virtual humans. This chapter presents an overview of relevant results in these areas and elaborates on the similarities and the differences between robotics and animation research.

2.1 Biomechanics

The biomechanics literature provides a variety of sophisticated models and data. A great body of work in this area involves producing anthropometric parameters for the average human. Such information includes static parameters such as the dimensions and the absolute and relative weights of the body parts of the average human, Winter [109]. Anthropometry is also concerned with dymamic parameters such as the maximum and minimum torques that the average human can exert with his/her muscles. Komura et al. [59] propose a method for the calculation of the maximal acceleration and force that a simulated model exhibits during arbitrary motions. This method can be used to enforce physical limits on the accelerations and forces associated with simulated motions.

The biomechanics literature is also a useful source of predictive models for specific motions, typically based on experimental data supplemented by careful analysis. These models target applications such as medical diagnosis, the understanding and treatment of motor control problems, the analysis of accidents and disabilities, and high-performance athletics. Computer simulation is becoming an increasingly useful tool in this domain as the motion models evolve to become more complex and comprehensive. Given the challenge of achieving high-fidelity motion models for individual motions, there have been fewer efforts towards integrated solutions applicable to multiple motions. The work of Pandy [82] is one such example.

The human body is capable of performing a wide range of motor control tasks, ranging from standing in place, to challenging athletic maneuvres such as a high bar kip. One of the most fundamental tasks that humans are able to perform is stand in place (quiet stance). Even such a simple task involves a number of subtle motor control issues. Fitzpatrick et al. [30] investigate the reflex control of postural sway during human bipedal stance. They find that reflex feedback related to ankle movement contributes significantly to maintaining stance and that much of the reflex response originates from lower limb mechanoreceptors that are stimulated by ankle rotation. Fitzpatrick et al. [32] discuss how the stiffness at the ankle joints is used as a response to gentle perturbations during standing. Fitzpatrick and McCloskey [31] compare the role of different sensor mechanisms that can be used by humans to detect postural sway during standing. They conclude that, during normal standing, proprioceptive input from the legs provides the most sensitive means of perceiving postural sway. Gatev et al. [34] investigate strategies that people use to maintaining balance during quiet stance and subject to gentle perturbations. They discover that subjects increased their use of the hip joints as the width of the stance became narrower. In addition, they find evidence which suggests that the slow and small sway that is present during quiet stance might be important to provide updated and appropriate sensory information helpful to standing balance.

Understanding the response of the human body to large external disturbances is very important for clinical studies and for identifying ways to prevent falls among the elderly. At the same time it is an important action for robotic and animation applications that involve autonomous agents. Large disturbances can induce significant velocity on the center of mass. Recent research work in biomechanics tries to understand the relationship between the velocity and the position of the center of mass and how it affects the response of a human subject. Pai and Patton [81] determine which velocity–position combinations a person can tolerate and still regain balance without initiating a fall. Their work employs a two segment inverted pendulum model and focuses on anterior movements. Pai and Iqbal [80] use a simple inverted pendulum model to compute similar feasible regions in the case of slipping on floors with varying friction and forced sliding.

When a person is unable to remain standing in place under the influence of an external disturbance, he/she implements a stepping behavior in an attempt to terminate the movement and regain balance. Romick-Allen and Schultz [93] report a variety of strategies that people use to maintain balance, including arm swing and stepping. Surprisingly, they conclude that human subjects standing on a moving platform, respond to an anterior acceleration of the platform with shoulder flexion that initially promotes rather than arrests the fall. Do et al. [24] investigate the biomechanics of balance recovery during induced forward falls focusing on the first step that subjects take to regain balance. This first step was characterized by two phases, a *preparation* phase and an *execution* phase. The preparation phase that ends with the toe-off precedes the actual step execution and its duration is invariant with respect to the initial conditions. Wu [114] studies the velocity characteristics of involuntary falls and concludes that during a fall the magnitudes of the horizontal and vertical components of the velocity of the center of mass increase simultaneously up to 2-3 times that of normal velocities.

When a stepping response is not sufficient to regain balance, human subjects employ a falling behavior which aims to cushion the impact with the ground. The behavior of choice depends on the direction of the fall, the age of the subjects, their athletic abilities and their personal preferences. Kroonenberg et al. [105] identifies the characteristics of voluntary falls from standing height focusing on sideways falls. They compute the velocity of the wrist and the hip just before impact and they investigate ways to decrease the severity of the impact. Smeesters [97] determines the fall direction and the location of the impact for various disturbances and gait speeds assuming passive simulated falling. One of her results shows that fainting during slow walking is more likely to result in an impact on the hip. In contrast, fast walking seems to prevent sideways falls.

The biomechanics of every day motions have been studied extensively for clinical and simulation purposes. Papa and Capozzo [85] investigate the strategies used in sit-to-stand motions using a telescopic inverted pendulum and for a variety of speeds and initial postures. Even a relatively simple motion such as the vertical jump requires a great deal of research before it can be fully understood and reproduced by simulation and automated control techniques, as studied by Pandy et al. [84], Pandy and Zajac [83], Spägele et al. [98] and others. McMahon [71] investigates the running gaits of bipeds and quadrupeds focusing on the effect of compliance. Athletic motions have been studied extensively for purposes of increasing performance and injury prevention. The kip motion, described in Bergemann [11] is a prime example of such an athletic maneuvre.

This literature survey is necessarily incomplete given the large volume of publications on the biomechanics of all types of motion.

2.2 Robotics

Robotics and animation research have a significant overlap. Robotics uses simulation and visualization of animated models to test control algorithms and proposed robotic structures before constructing the actual robots. Most of the techniques developed in one area can be used in the other and vice versa. Therefore the division between research results between the two areas is relatively arbitrary. This section focuses on results that have been used on actual robots and not on the robotic techniques that are more commonly presented as part of the animation literature. These will be addressed later. However, there are some differences between animation and robotics. The most important one is that real robots are associated with a number of real world constraints such as the following:

- *Power source.* Robots need a power source and therefore their design must include one.
- *Ground.* Dust, humidity etc. affect the friction coefficient of the ground which is not perfectly homogeneous in the first place.
- *Noise.* Feedback is based on sensors which have noise in their measurements.
- *Hardware defects, unknown or difficult–to–model parameters.*

Working in simulation allows us to make a wide range of assumptions and design decisions for the character and its environment that can facilitate our results. In contrast, robots have to function in the real world over which we have limited control. In addition it is much easier to tune or alter a simulated character or its environment than a robot which takes a great amount of time and effort to assemble.

Robotics research has made remarkable progress in the successful design of a variety of legged robots. Some of the better known examples include Raibert [88] and, more recently, anthropomorphic bipedal robots such as the Honda [65], the Sarcos [55] and the Sony [66] robots. Despite their limited motion repertoires and rather deliberate movements, these robotic systems are truly engineering marvels.

Walking gaits are the focus of most research work involving humanoid and animal-like robots. The Honda robot is capable of walking, turning, ascending and descending stairs and a few other simple maneuvres in controlled environments. The Massachusetts Institute of Technology Leg Lab has produced a variety of bipedal robots such as Troody the dinosaur [60].

Troody, the Honda and the Sony robots are capable of performing relatively slow walking gaits. Gienger et al. [35] propose a bipedal robot model capable of jogging. Their controller is based on linear feedback and uses lookup tables for the optimal positioning of the feet. Inoue et al. [56] present a method that stabilizes a humanoid robot in mobile manipulation. As a result the humanoid robot autonomously steps or keeps standing, coordinating with the arm motion to achieve a position that maximizes its stability along with its ability to perform a task. Huang et al. [51] present an interesting method for bipedal robot walking over varying terrain and under external disturbances. They combine an off-line generated walking pattern with real-time modification to produce a robust planar walk that can adapt to changes in the ground properties and external disturbances. Chew and Pratt [19] investigate the use of machine learning in a bipedal walk. In particular, their method uses reinforcement learning to learn the positioning of the swing leg in the sagittal plane. In addition, they consider the balance on the sagital plane separately from the frontal one and they treat them separately. This allows existing planar walking algorithms to be combined with their balancing algorithm for the frontal plane.

While most of the research on humanoid robots focuses on particular motor tasks such as walking, a significant amount of research in robotics investigates ways to produce non-humanoid robots with complex behaviors. By using robots that are inherently easier to move about such as wheeled robots researchers can focus on the problem of integrating sensor information and behavior patterns to develop robots that are capable of performing complex tasks. An interesting example is the RoboCup project that involves robots competing in soccer [92]. Arkin [2] provides a good summary of behavioral architectures explored in the context of robotics. Among them, perhaps the most relevant to our work is the *subsumption* architecture proposed by Brooks [12]. The subsumption architecture advocates the simultaneous existence of prioritized modular behaviors. Behaviors are independent and there is very little communication between them. Each behavior keeps its own model of the world and has its own goals. Most of the behaviors are based on a stimulus-response principle. More complex behaviors subsume lower ones and they can suppress them when appropriate. The lower behaviors have no knowledge of the higher ones, allowing complex behaviors to be constructed incrementally. The subsumption architecture has been used successfully on variety of robots [2].

Burridge et al. [15] propose a sequential behavior composition method based on the funneling approach, where each behavior brings the system within the feasible region of another behavior until a goal is reached. Behaviors learn their feasible region through a sampling and learning approach. Learning the feasible region is similar to the approach we use to automatically compute the pre-conditions of our controllers. Using their technique they develop a robotic paddle that can dynamically juggle a ball. The practicality of extending this method to high-DOF dynamic models of human motions is unclear.

2.3 Computer Animation

Computer animation is to a large extent unencumbered by the exacting fidelity requirements of biomechanical models and the mechanical limitations of robotic systems. This has spawned a great variety of kinematic and dynamic models for character motion [4, 5, 16]. Motion capture solutions are very popular because they can accurately capture all the subtleties of the human motion. However, motion capture methods have several limitations:

- They depend greatly on a specific subject and they cannot be easily generalized for other models.

- Not everything can be motion captured. Some examples include dangerous stunts and dinosaur motions.
- Kinematic motion cannot be easily used for interactive characters that must support physical interaction such as collisions.

Blending, warping and other motion curve manipulation techniques, Witkin [111], Bruderlin [13], Gleicher [36], have been used to alter original motion capture data. Although these techniques produce satisfactory results in the short term, physics-based approaches reveal more about the physics, planning, and control of such motions and they therefore serve as a basis for more general solutions. Dynamically simulated characters were first proposed over 15 years ago [3, 108] and since then have progressed in sophistication in a variety of directions. Controllers have been successfully designed for specific human motions. Hodgins et al. [46] implement a variety of athletic motions such as running, vaulting, and cycling. Their controllers are based on finite state machines and use feedback to properly balance the character. Wooten and Hodgins [112] implement a virtual diver that is capable of performing elaborate aerial maneuvres. A planar walking controller capable of handling variable terrain is implemented by van de Panne et al. [104]. The controller interpolates between pre-computed lookup tables that contain optimal control solutions. An elaborate walking controller for a full 3D human model is developed by Laszlo et al. [62]. The controller is based on a local stabilization technique that adds closed–loop feedback to open–loop periodic motions. Grzeszczuk and Terzopoulos [42] develop a neural network controller that can learn the physics of a motion and then produce a relatively accurate kinematic version of it.

Motion synthesis for complex models is a very difficult task. It is therefore natural that a great amount of research investigates ways to automatically synthesize motions. The spacetime constraints method proposed by Witkin and Kass [110] and its variants (Cohen [21], Liu [63]) allows the animator to specify what the simulated character should do and to a limited extent how. An optimization process computes the trajectories of the character's degrees of freedom such that the user–specified constraints are satisfied. The physical laws participate in the optimization process as soft constraints. The computational expense of the optimization process restricts these methods to relatively simple character models. Popovic et al. [87] present a technique that allows the interactive manipulation of physics-based simulation of rigid bodies. The user can interactively specify constraints over the motion of the participating bodies or simply drag them to desired positions. In response, the system computes the required physical parameters and the resulting trajectories. This technique is limited to the simple objects and it does not seem to scale well for complex articulated characters. Chenney and Forsyth [18] propose a method for controlling the motion of multiple bodies in collision intensive situations. Their method is based on sampling the solution space for plausible solutions that satisfy user specified constraints. This method is also limited to simple bodies.

Instead of synthesizing trajectories, a different approach is to synthesize controllers which coordinate the motors of the simulated characters to produce specific motions. Controller-based methods provide a convenient and realistic separation between the control and the dynamics of the character. Controllers represented as finite state machines are automatically produced using a stochastic generate–and–test approach by van de Panne et al. [102]. Similarly, controllers that consist of networks of sensors and actuators are automatically produced by van de Panne and Fiume [100]. Ngo and Marks [79], use a genetic approach to automatically evolve controllers that are represented by sets of stimulus-response rules. Grzeszczuk and Terzopoulos [41] using a stochastic optimization process to teach fish models how to swim. Automatic motion synthesis based on generate–and–test methods cannot be used for complex characters and highly dynamic motions. The search space is so large that it cannot be handled by the current methods. Laszlo et al. [61] propose an interactive method of synthesizing controllers for arbitrary characters including 2D humanoid models. This method implements a *performance-based* approach to physics-based control that exploits the user's intuition and knowledge about motion planning.

The differences between motion synthesis methods that are based on motion capture techniques and those that are based on physical simulation lead researchers to investigate hybrid methods that combine the advantages of both. A nice overview of such results can be found in the SIGGRAPH 2000 course notes [48]. Playter [48] explains how Boston Dynamics Inc. [53] uses a combined motion capture and physics-based control method to develop a planar runner. The controller is based on Raibert's [88] state machine. A motion playback module is used to synchronize the reference trajectories with the actual motion. Posture, speed and hopping control are used to maintain balance. The resulting trajectories are used to drive the servo-mechanisms at the joints of the character.

2.4 Controller composition

Dynamically simulated articulated characters equipped with an integrated, wide-ranging repertoire of motor skills currently remain an unachieved goal. Some positive steps in this direction are evident, however. Tu and Terzopoulos [99] develop an integrated repertoire of motor controllers for biomechanically animated fish. Grzeszczuk and Terzopoulos [41] develop dynamic controllers that can be sequenced using an overlapping and blending technique. Funge et al. [33] use a similar method to compose swim controllers for a "merperson." A higher level planning module based on cognitive modeling and predicate logic determines automatically the sequencing of the swim controllers. The work by van de Panne et al. [103] proposes a methodology for controller design and integration applicable to simple figures. Finite state machine controllers that implement gaits for simple figures can be parameterized producing variations of the original gaits, van de Panne et al. [101]. Linear combinations of these parameterized controllers produce intermediate variations and transitions between gaits. The work of Wooten [113] is the most relevant as an example of a sequence of successive transitions between several controllers for human motions such as leaping, tumbling, landing, and balancing. The controllers are manually sequenced and transitions happen when the end state of one controller is within the region of starting states of the next controller in the sequence. Hodgins [44] develops algorithms for walk-to-run and run-to-walk transitions for a 2D biped. However, it is seems that the proposed methods cannot be applied to the 3D case. Maes and Brooks [68] propose a method for coordinating behaviors that know when they should be active. Using a method similar to reinforcement learning a six legged robot learns to coordinate its legs to produce a stable walk. Two touch sensors and a sensor that measures the horizontal distance traveled provide positive and negative feedback based on which each behavior learns when it should be active. There is one behavior associated with each of the 12 degrees of freedom of the legs. A *digital biomechanics laboratory* is proposed by Boston Dynamics, Inc. [53] as a tool for simulating a wide range of human motion. This currently remains ambitious work in progress.

The concept of pre-conditions, which is central to our work, has been used before in the control and artificial intelligence literature. Precedents of pre-condition evaluation exist in terms of estimating the stability of a given system or determining the region of attraction of a dynamical system. Such regions can be computed analytically for linear systems. For more complex system this is usually done through extensive and exhausting point sampling most often to create two dimensional illustrations which depict appropriate regions of stability or attraction.

Our work is aimed at creating dynamic human characters with broadly integrated action repertoires. Unlike previous work focusing on specific athletic movements, our methodology is to begin with a core set of simple actions, including balancing, small steps, falling reactions, recovery from falls, standing up from a chair, and others.

2.5 Simulated control systems

There is a variety of systems that have been developed to facilitate the control of articulated figures in the scope of robotics and animation. Reichler and Delcomyn [91] present a nice overview of the different methods and software available for robotics research.

2.5.1 Commercial animation software

There is a great number of commercial animation packages such as Maya, 3D Studio Max, Poser 4, Rhino, Houdini, SoftImage etc. Some of them are script based or implement plugin APIs, *Falling Bodies*[52], Mathengine [70], Havok [54]. We believe that a common research tool based on a commercial package is not likely to happen because commercial software presents the following problems:

- *Price.* High-end software packages can be too pricey for students, researchers and hobbyists.
- *Lack of generality.* Software packages may incorporate the state-of-the-art, but often do not allow modifications that are essential for research that goes beyond the state-of-the-art.
- *Lack of portability.* Most commercial products run on specific platforms and porting is under the control of the company.
- *Lack of industry standard.* None of the commercial products is currently the industry standard.
- *Closed environments.* Some commercial packages come with their own simulator, integration method, or collision detection technique which cannot be changed.
- *Exclusive simulator.* To our knowledge most animation packages do not allow multiple simulators.
- *Technical issues.* Some commercial packages consume a lot of resources and can run only on high-end machines.

2.5.2 Robotics

Reich et al. [90] implement an environment for complex biomechanical simulation. Apparently their implementation is focused on specific biomechanical models and it does not provide APIs that can be used to extend the functionality of the system.

Reichler and Delcomyn [91] present an interesting system for the dynamic simulation of legged robots. Their system uses a high level configuration language that acts as a bridge between simulation components. It can be used to efficiently describe robots, their actuators, sensors and controllers in a uniform fashion. A generalized referencing scheme allows control structures and user-interface elements to communicate with each other. Under this scheme most components are black boxes with inputs and outputs that can be wired to each other. Users can extend the software using C++ programming. The aim of this work is to provide a common system for the rapid prototyping of controllers for legged robots and it is limited to a specific simulator, although the high level language could be used on top of other simulators. The use of Euler integration is a potentially limiting factor. It seems that this system is geared towards simpler robots and to our knowledge it has not been used for human locomotion.

Dynamic animation tools are available mostly in the form of libraries that provide the equation of motion for articulated figures. Such systems focus on providing efficient dynamic code and they can be turned into plug-in simulators for our system. Mathengine [70], Havok [54] and SD/FAST [49] are examples of such dynamic tools.

The Robotics Toolkit for Matlab, Corke [22] and the Robotica for Mathematica, Nethery [76] examples of toolkits that run on top of commercial mathematical packages. They provide functions and objects that can be assembled together to a construct a simulation experiment. However, they do not seem to provide an interface for attaching sensors or actuators.

Our system, presented in Section 6, aims to provide a free and open tool that can support a wide range of environments through generic APIs for animation research. To our knowledge such a system does not exist.

Chapter 3

Composition Framework

The work in this chapter attempts to answer the following question: Given two controllers, one designed for walking by a group A and one designed for standing-in-place by group B, how can we produce a composite controller such that the associated simulated character has the combined functionality, i.e. walk and stand-in-place? Composing controllers that have been designed separately is an interesting issue in animation and robotics. It is also a difficult problem to solve.

3.1 Composing controllers

We assume that a character is equipped with a number of controllers and each controller is capable of performing a specific motor task. The problems are (a) how to decide which controller should be applied at each point in time and (b) each time a new controller is chosen to assume control, how do we transition to it successfully. The first problem can potentially be solved in a variety of ways, ranging from randomly picking controllers to elaborate planning processes. Approaches to the second problem, transitioning successfully to a new controller can be classified in three categories as described in Wooten [113]. Assuming n different controllers the most complete method requires n^2 transition controllers that can take the system from the end state of one controller to the start state of another controller. A simpler solution is to implement $2n$ transitions: n from every controller to a common intermediate state and n transitions from the intermediate state to each of the n controllers. Alternatively, we can eliminate transitions altogether by making sure that the end state of a controller falls within the start states of another controller. However, the last two methods may not be appropriate for all cases. The above discussion and most of the previous work in the area assumes that transitions happen at specific entry points, mainly from the "end" state of one controller to the "start" state of another controller. However, this is not the general case. Some controllers such as those designed for walking and running, do not have well-defined "start" and "end" states. In dynamic environments unexpected things can happen that do not let a controller complete its operation and require immediate transition to other controllers. Thus the number of transitions can be generally higher than n^2. Figure 3.1, shows possible transitions between Controller A and Controller B within an abstraction of the character's state space. Two of the transitions are realized by separate controllers. When the character is in a state within the region of overlap, then we can immediately switch to either controller. Our system allows for such dynamic transitions as we will see below. Section 3.8 explains in detail how our framework deals with transitions between controllers.

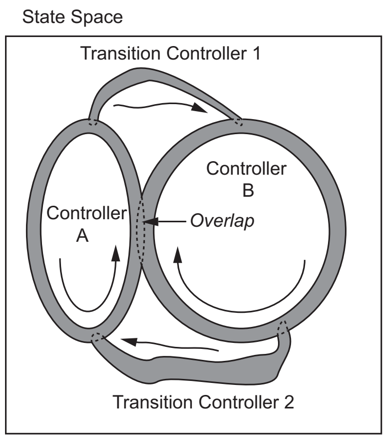


Figure 3.1: An abstract visualization of potential transitions between controllers for walking and running.

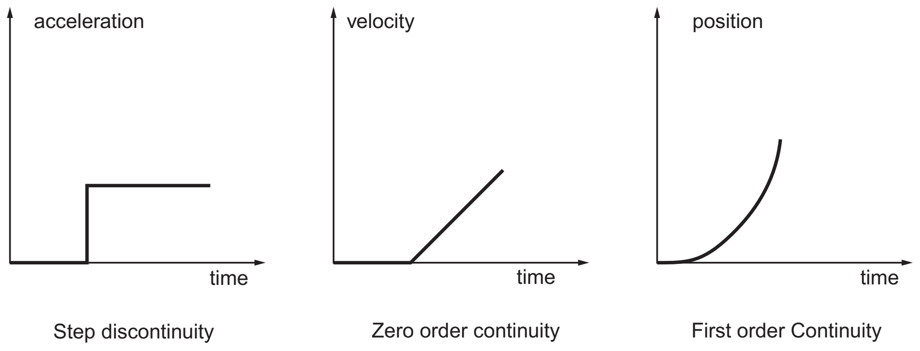


Figure 3.2: Degrees of continuity.

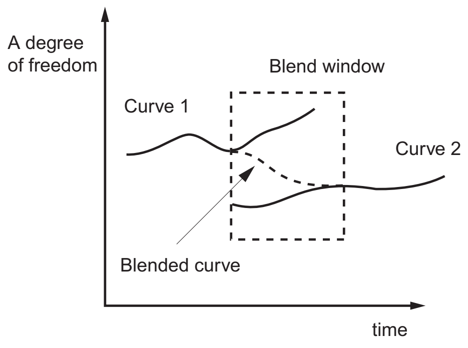


Figure 3.3: Motion curve blending.

An important technical difference exists between composing motions kinematically and dynamically. In the kinematic case, the primitives of motion are curves that specify the position of the character's degrees of freedom in time. The composition process must make sure that these primitives are composed smoothly and that both the velocity and the position of the character's degrees of freedom are continuous. In the physics-based (dynamic) case, the primitives of motion are curves that specify the forces that the character's actuators (muscles) exert over time. When we switch between controllers, in the worst case, there is a finite discontinuity (step) in the forces, which translates to a step discontinuity in the acceleration of the character's degrees of freedom. Since acceleration is the first and second derivative of the velocity and the position respectively, we are guaranteed that velocity and position are respectively zero and first order continuous, see Figure 3.2. Our composition framework can work with both kinematic and dynamic controllers. However, unless the kinematic controllers themselves make sure that they take over smoothly, we would have to implement an interpolation scheme that blends between kinematic controllers. Figure 3.3 depicts the three different degrees of continuity along with a typical blending scheme for control curves.

From the above discussion, it is clear that composing controllers requires a module that acts as an arbitrator between the available specialist controllers. The resulting composed controller, which includes the arbitrator, must be able to activate the appropriate controller at each point in time and choose among controllers that can possibly do the same thing but in a different way, or in a different style. For example, there is more than one way to get up from a prone position; these motions can be radically different and can be implemented by separate specialist controllers. The arbitrator must be able to resolve such cases.

3.2 Our composition framework

Typically, a composition scheme has two levels as shown in Figure 3.4. The lower level consists of the available controllers that implement specific motor control skills. The upper level coordinates the use of the lower level appropriately. The main issue in such a scheme has to do with the way the intelligence of the system is distributed between the two levels. The flow of information and the interaction between the two levels determines the properties of our scheme such as scalability, expandability and ease of use. In other words, the composition scheme can be centralized or distributed in varying degrees depending on the desired properties. Our

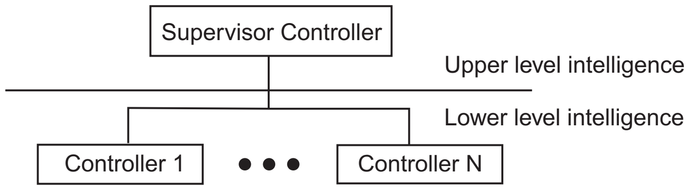


Figure 3.4: Two level composition scheme.

scheme is largely distributed. We choose to push the intelligence to the lower level in a way that achieves the following properties:

- *Scalability.* Our framework can handle a large number of low level controllers.
- *Expandability.* The system can incorporate new low level controllers at run time.
- *Simplicity.* The composition method is straightforward and easy to implement. It does not appreciably burden the controller design task.

- *Generality.* The composition scheme does not restrict the design of individual controllers. Each controller can be as primitive or as sophisticated as its designer wishes.

These properties are important for any composition scheme that aims to become a unifying framework for work in this area. Practitioners need the freedom to design controllers with the desired complexity and with the preferred architecture. If a system restricts either the complexity or the way a controller operates, then practitioners may not use it.

Our framework follows the two level composition scheme shown in Figure 3.4. Within this framework we consider individual controllers as black boxes which are managed by a simple supervisor controller. When no controller is active, the supervisor polls the pool of controllers querying each whether it can handle the transition of the dynamic character from its current state to the desired goal state. Individual controllers return an integer confidence/suitability score when queried in order to bid for becoming the active controller, as detailed in Section 3.7.

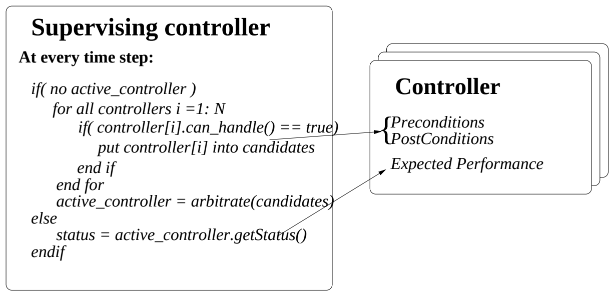
3.3 Controller abstraction

A controller within the pool of available controllers can be as simple as a constant force, or as complex as a structured hierarchy of multiple levels of control abstraction. For example, as more controllers are added to the system, we may wish to group all the walking and running controllers together into a cluster that can be treated as one controller. This grouping may happen at various levels, where the first level corresponds to the parameterization of controllers. In any case, instead of trying to design a single super-controller, knowledge about how to control the dynamic character is left for designers to encapsulate within their control algorithms, since they are best equipped to determine the conditions under which their algorithms are able to operate.

Regardless of the encapsulation, our composition method requires the definition of *pre-conditions*, *post-conditions* and *expected performance* for each controller. Pre-conditions are a set of conditions over the state of the character and the environment. If these conditions are met then the controller can operate and possibly enable the character to satisfy the post-conditions. The post-conditions define a region of final states that the character can reach after the successful execution of the controller. In other words the controller realizes a mapping between a domain of input states to a range of output states for the character. Because of unexpected changes in the environment this mapping may not always succeed, which motivates the notion of expected performance. The controller should be able to evaluate its performance in order to detect failure at any point during its operation. To do this, the controller must at all times have knowledge of the current and expected state of the character or the environment.

Defining the pre-conditions, post-conditions, and expected performance for complex characters, motions, and environments is not a straightforward task. However, we believe that the effort required to generate these specifications is a fair and necessary price to pay to achieve the benefits of composability. Controllers that adhere to these specifications can form a pool of available controllers managed by the supervising controller. Fig. 3.5 presents an overview of how the supervising controller works and its interaction with the individual controllers at every time step of the simulation.

Before we elaborate on pre-conditions, post-conditions, and expected performance in subsequent sections, let us define the following quantities and symbols: The *state* $\mathbf{q} = [\mathbf{x}\ \dot{\mathbf{x}}]'$ of a figure is the vector of generalized positions $\mathbf{x}$ and velocities $\dot{\mathbf{x}}$, where the dot indicates a time derivative. The position and velocity of the center of mass are denoted as $\mathbf{c}$ and $\dot{\mathbf{c}}$ respectively.



```
if( no active_controller )
    for all controllers i =1: N
        if( controller[i].can_handle() == true)
            put controller[i] into candidates
        end if
    end for
    active_controller = arbitrate(candidates)
else
    status = active_controller.getStatus()
endif
```

Figure 3.5: Controller selection and arbitration during simulation.

The *base of support* of a figure (often called the *support polygon*) is denoted as $\mathcal{S}$. It is represented by a polygon that surrounds the foot or feet that are in contact with the ground at any given time.

3.4 Pre-conditions

In general, pre-conditions are relationships and constraints involving a number of different parameters. We have used the following parameters in our work:

- The initial state $\mathbf{q_i}$ of the figure. Most of our controllers can operate within a small region of the state space which we denote $\mathcal{R}(\mathbf{q}_i)$.
- Environmental parameters. These include the contact points between the character and the ground, as well as the normal of the ground and the amount of friction at the contact points. In the following we denote conditions (generally indicated by the letter C) on the environment parameters as C_e.
- The balance of the figure. Usually, this is indicated by the relative position and velocity between the figure's center of mass $\mathbf{c}$ and the base of support. Typically, if the projection of $\mathbf{c}$ along the gravity vector $\mathbf{g}$ does not intersect the base of support $\mathcal{S}$, the figure is considered to be unbalanced. We denote the balance conditions as $C_b(\mathcal{S}, \mathbf{g}, \mathbf{c}, \dot{\mathbf{c}})$.
- A target state $\mathbf{q}_t$, or in general a target region of the state space $\mathcal{R}(\mathbf{q}_t)$, which can be provided by the user.

Pre-conditions consist of unions of instances of the above conditions and are denoted

$$\mathcal{P} = C(\mathcal{R}(\mathbf{q}_i), \mathcal{R}(\mathbf{q}_t), C_b, C_e). \tag{3.1}$$

The determination of pre-conditions is crucial to the success of our composition framework and will be examined in detail in Section 3.9.

3.5 Post-conditions

Successful operation of a controller brings the character from an initial state, as defined by the pre-conditions, to a desired state or a desired region $\mathcal{R}(\mathbf{q}_o)$ in the state space. This region along with balance C_b and possibly environmental constraints C_e form the post-conditions of a controller:

$$\mathcal{O} = C(\mathcal{R}(\mathbf{q}_o), C_b, C_e). \tag{3.2}$$

3.6 Expected performance

Our framework permits the automatic selection of the appropriate controller based on the information provided by the controllers themselves. Only the individual controllers can detect whether they are operating normally or they are going to fail. Failure in our case means that the controller cannot meet its post-conditions $\mathcal{O}$. A sudden change in the environment or badly designed pre-conditions can make the controller fail. The sooner a controller can detect failure the sooner another more appropriate controller can take over. This is important for making a character behave naturally. For example, the character should not attempt to continue a walking gait if it has lost its balance and it is falling. In our implementation, the expected performance $\mathcal{E}$ consists of expressions similar to those of the pre-conditions $\mathcal{P}$. In particular if the controller successfully completes its task in the time interval $[t_1, t_2]$ then:

$$\mathcal{E}(t_1) \in \mathcal{P} \quad \texttt{and} \quad \mathcal{E}(t_2) \in \mathcal{O}.$$

It is worth noting that the expected performance conditions vary for different parts of the motion. For example, the first phase of a motion might be balanced while the second phase may be unbalanced.

3.7 Arbitration

There is often the case that more than one controllers are suitable for the current state of the character. For example, there are many ways to get up from a prone position. In addition, a controller can often handle the current state of the character but have a different goal than the one desired. For example, upper stance is suitable for both the balance and the walk controller. However, the controllers have clearly different goals one of which might match the goal specified by the user. Our supervisor controller has no knowledge of what the controllers can do and therefore cannot decide what to do in such case on its own. It is up to the individual controllers to guide the supervisor controller towards this decision through some simple communication. In the current implementation, when a controller is asked to bid for control based on its pre-conditions it returns a confidence score equivalently encoding suitability or priority which is an integer number in the following ranges: 0 if its pre-conditions are not met, $[1 - 10]$ if it can handle the current state only and $[11 - 20]$ if it can handle the current state and achieve the current goal. Under this scheme controllers that can meet the given goal take precedence over controllers that can just do something for the current state. The exact value of the confidence number in a given range is arbitrarily defined. A heuristic rule that we follow is to give low confidence numbers to controllers that are general and can handle a large number of states. For example, the fall controller can handle a larger number of falling states and has lower priority compared to the protective-step controller. Thus, when the protective-step controller thinks it can handle the current state it takes precedence over the fall controller. For controllers designed by different developers, priorities can potentially be assigned or remapped to different ranges as desired.

Considering the nature of a motion can help assign an appropriate priority to the associated controller. For our purposes we classify human motions in the following categories:

- *Reactions.* They refer to reflex or largely unconscious reactions such as attempting to maintain balance.
- *Actions.* These are motions that we perform only consciously. They often have a clear goal.

- *Quiscent states.* Standing in place or sitting on a chair is more of a state than an action. Such motions do not have a specific duration or a clear end-goal.

Actions typically have higher priority than reactions or states. However, the associated action controllers bid for control only when the user specifies a target state that falls within their pre-conditions. Most controllers associated with states bid for control even when the goal is not matching their own. For example, if the user requests that the character jumps in the air and there is no controller that can perform the jump it makes sense that the character continues to sit or stand. Although typically they do not conflict with each other, reaction controllers have higher priority than the state ones. The above categorization of motions is arbitrary and only serves as a reference. Controller developers can assign priorities according to their own intentions.

3.8 Transitions

Transitions between controllers are not explicitly modeled as they would be in a finite state machine. They occur implicitly in response to the evolution of the motion over time, as the system state traverses the 'regions-of-competency' of the various controllers. Nevertheless, typical patterns of controller activation occur given that most controllers are designed for specific situations. Fig. 3.6 shows the family of controllers designed for the 3D dynamic character and their typical transition patterns. For example, the controllers and transitions used in achieving the motion shown in Fig. 1.3 is given by balance → fall → default → rollover → prone-to-standing → balance. Fig. 3.7 similarly shows the family of controllers designed for the 2D dynamic character and their typical transition patterns. Note that not all possible transitions are shown in either of Figs. 3.6 and 3.7. For example, the prone-to-standing → fall transition can occur if the figure is given a sufficiently strong push while rising. Most of the transitions which are not shown but are still practically feasible are of this nature, dealing with falling behaviors. It is worthwhile noting that the fall controller always responds to the specific direction of the current fall.

Any transition involves one controller being deactivated and another being activated. A controller can become deactivated (and thereby eliciting a transition) for one of three reasons. First, it may itself give up control by declaring success in reaching its post-condition, as is the case for a standup controller which has successfully returned the character to a standing position. Second, user intervention may elicit a transition. The controllers designed for sitting or balanced standing will retain control until intervention by a user (or by a higher level planner) forces a desired transition. Thus, when the 2D character is balanced a user-driven process must choose among the next plausible actions, namely one of *sit*, *walk*, or *dive* (see Fig. 3.7). Third, a controller may detect failure, as will be the case for unpredictable events such as a push or an unforeseen obstacle causing a trip. The transitions in Figs. 3.6 and 3.7 are labeled according to the type of controller deactivation which typically spawn the given transition patterns.

3.9 Determining pre-conditions

For controllers associated with complex dynamic characters, quantifying the exact region of the state space and the general conditions that determine success or failure of the controller is in general a non-trivial matter. This is due to the high dimensionality of the state space and the irregular shape that the pre-conditions region has. In this thesis, we address this problem

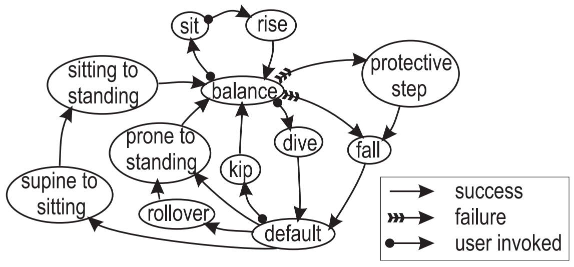


Figure 3.6: Controllers and typical transitions for 3D figure

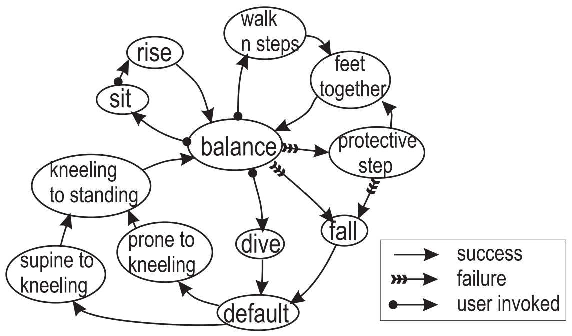


Figure 3.7: Controllers and typical transitions for 2D figure

via manual and automatic approaches, in turn. The manual approach allows the designer to incorporate his or her knowledge within the controller, whereas the automatic approach is based on machine learning techniques.

With both approaches it is generally impossible to compute a strictly conservative approximation of the pre-conditions. Either the time required to compute the approximation is prohibitive or the size of the model produced impossible to store and work with. In general both methods will result in an approximation of the pre-conditions. There are certain implications of having pre-conditions that are overly generous. Controllers will be invoked only to fail often. Conversely, an overly conservative approximation means that a controller will be chosen less often than it perhaps should be, and therefore it will be more difficult to design other controllers to work in conjunction with it. We aim to build reasonable approximations of the pre-conditions. Some small underestimation will not cause significant problems while some small overestimations will cause the controller to fail to execute its task in rare circumstances. With a sufficiently large pool of controllers, the arbitration scheme can provide reasonable means to recover once the failure occurs.

3.10 Manual approach

For certain cases, suitable pre-conditions for specific controllers may be found in the biomechanics literature [24, 81]. For example Pai and Patton [81] present a comprehensive study of balance in the sagittal plane and identify the conditions under which a human can compensate for postural disturbances and maintain balance without stepping. Certain controllers function as intermediate stages between other controllers. If controller B is the intermediate step between A and C then the post-conditions of A must be a subset of the pre-conditions of B and similarly the post-conditions of B must be subset of the pre-conditions of C. In general, to ensure that controller A can transition to controller B, we have to make sure that the post-conditions of A are a subset of the pre-conditions of B. If this is not the case then there are three ways to remedy the problem:

1. Refine the control of A to enforce a tighter set of post-conditions.
2. Refine the control of B to enlarge its pre-conditions such that they include the post-conditions of A.
3. Do both.

In some cases the pre-conditions are computed by manual experimentation. For example a simple balance controller for a standing human model based on an inverted pendulum model, Fitzpatrick [32], has intrinsic stability that can tolerate small disturbances. After the controller has been designed, repeated testing under disturbances of increasing magnitude can yield an approximation of the pre-conditions and the post-conditions.

In any case, the designer of a controller presumably understands the way the controller operates, and thus is able to provide high level conditions on its success or failure. For example, the designer of a walking controller knows if the controller can operate when the walking surface has minimal friction properties. Also, human motion is governed by notions such as comfort, and only the designer can take this into account. For example, people that are pushed while standing might take a step instead of employing an equally successful inverted pendulum approach just because it may be more comfortable to do so. Similarly, reactions to slipping and unbalance and protective behaviors are largely age dependent [50].

Alternatively, the expected performance can be computed by sampling the trajectory of key parts of the character during a test run of the controller. During its operation the controller can verify that the current trajectories match the expected motion to within some allowable margin of error. However, choosing an acceptable error margin is not trivial. We did not use this method in our experiments.

Chapter 4 discusses an alternative way to compute the pre-conditions of controllers. The proposed method is based on point sampling the state space and it employs a machine learning method to produce an approximation of the pre-conditions region. Our controllers and their manual pre-conditions are described in Chapter 5. Chapter 7 discusses our results using both manual and learned pre-conditions.

3.11 Discussion

The selection and arbitration processes are distributed to the specialist controllers in our framework and this has several consequences. There is no “brain” in the system that makes decisions in a centralized way. Most of the processing and the intelligence of the systems resides in the individual controllers that come armed with the knowledge of what they can do and when they can do it. This feature is both an advantage and a limitation. The advantage comes from the fact that controllers can be easily added to the system, even at run time. In contrast with previous work, our composition framework does not know what the controllers can do; it finds out dynamically and as the need arises. However, at some point our system will need a centralized brain or planner that will decide what is the desired goal of the character at each point in time. Although a motion planner is beyond the scope of this thesis, the current system has many features that would be useful for building a motion planner. A more elaborate discussion on the limitations of our system is presented in Chapter 8.

Chapter 4

Learning pre-conditions

In this chapter, we introduce a semi-automatic, machine learning approach to determining pre-conditions, which is based on systematically sampling the performance of controllers. Our method uses a machine learning algorithm attributed to Vapnik [107] known as *Support Vector Machines* (SVMs), which has recently attracted much attention, since in most cases the performance of SVMs matches or exceeds that of competing methods. The next section discusses background information on machine learning which is based on [20].

4.1 Machine learning

For a variety of complex problems, there is no mathematical model that describes how to compute a desired output from a set of inputs. In addition, in some cases that computation may be very expensive to perform. An alternative approach to solve these problems is to make the computer learn the input/output relationship from examples. This is the same approach used to teach children to distinguish sports cars from the rest [20]. By seeing a sufficient number of sports cars, they learn to recognize them without being given a concrete definition of "sportiness." The approach of using examples to create programs is called the *learning methodology.* When the examples are input/output pairs, the learning methodology is called *supervised.*

If no noise is present, the pairs of input/output are samples of a function mapping between the input and the output. The learning approach aims to provide an approximation of that function. The function we attempt to model is called the *target* function and the output of the learning method is the *solution* of the problem. The solution is usually chosen from a family of known functions which form the *hypothesis space.* For example the hypothesis space might consist of the polynomials of third degree. The hypothesis space is one of the most important ingredients of the learning methodology. The algorithm that takes as input the training data and selects one function from the hypothesis space is called the *learning algorithm.* When the function we wish to model is a binary function the problem is called *binary classification.* For the case of classification the solution is called the *decision function.* The ability of the machine learning methods to model the training data is called *fitting* while modeling data that is not part of the training set is called *generalization.*

Machine learning is a very attractive approach that can handle a wide variety of problems. The use of examples for learning is a very familiar and intuitive approach that can avoid expensive computations. At the expense of collecting the training data machine learning can avoid expensive modeling computations that might be required by other approaches. However,

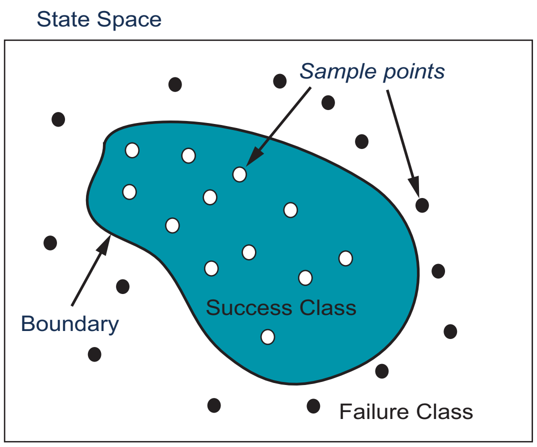


Figure 4.1: Training set and actual boundary for a 2D problem.

as with every method, machine learning has a number of drawbacks:

- *Efficiency.* A number of issues might affect the efficiency of a learning algorithm, for example the problem of local minima.
- *Size of output.* The size of the output might become too large in some cases, and therefore impractical to use.
- *Overfitting.* If the training set contains too few examples, a rich hypothesis space might lead to poor generalization.
- *Parameter tuning.* Learning algorithms often depend on a variety of parameters whose value is chosen based on heuristic criteria. Thus, the associated systems might be hard to use and potentially unreliable.

4.2 Learning the pre-conditions as a machine learning problem

The focus of this chapter is an automated method of approximating the region in the state space of the character for which a given controller can successfully operate. We call this region the *region of competence*. The state space of a humanoid character is a bounded, continuous space of high dimension. Modeling a controller's region of competence within this space generally cannot be done in an analytic way. We therefore formulate the problem as a machine learning problem. For each controller, we need to train a machine that when given an arbitrary initial state of the character, will predict whether the controller will succeed or fail. This is a typical classification problem. For any given controller, the state space of the character can be separated into two classes, one that leads to success and one that leads to failure. We want to train a machine that can assign an arbitrary state to one of these classes. To employ classification techniques we first have to produce a sufficiently large set of known examples, called the *training set* which will be used to train a classifier. Figure 4.1 depicts a two dimensional projection of the state space, the two classes of states (success, failure), the boundary between the classes and sample states that are part of a training set. The next sections describe how we produce the training set and our choice of classification method.

4.3 Choosing a classification method

There is a wide variety of machine learning techniques that can provide a classifier for our application. Unfortunately, there is no one method that is considered the best one for all cases. Each method has different properties that make it suitable for a particular kind of application. Knowing the domain of an application, that is knowing the form of the problem we are trying to solve using machine learning, is probably the best guide in choosing a suitable machine learning technique. In our case the main properties and requirements of our application are:

- *Off-line learning.* We can produce our training set and therefore train our classifier in advance.
- *Size of the training set.* We need to provide a large number of training examples in order to include enough samples around the boundary between the two classes. The classification algorithm should be able to handle a large number of high dimensional input examples.
- *Response time.* The classifier will be part of an interactive system which means that the response time of the classifier must be short.
- *Size.* There will be one classifier for each controller present in the system, thus the memory signature of the classifier must be relatively small.
- *Ease of use.* The classification algorithm must have intuitive parameters that we can use to adjust its behavior.
- *Lack of structure.* We do not have an intuition about the shape of the boundary. Our experiments have shown that it varies radically among controllers and that, generally, it has an irregular shape.
- *Generalization.* We need a classifier with strong generalization power and therefore one that has a sufficiently small hypothesis space.

The classification algorithm we have chosen to use satisfies the above requirements and exploits to some extent the properties of our particular class of problems. We present this algorithm in the next section.

4.4 Support Vector Machines

SVMs are a method for fitting functions to sets of labeled training data. The functions can be general regression functions or they can be classification functions. In our application, we use simple classification functions with binary outputs which encode the success or failure of a controller.

Burges [14] provides an excellent tutorial on SVMs and Christanini and Shawe-Taylor [20] provide an extensive description of SVMs and other kernel methods. Mathematically, we are given l observations, each consisting of an d-dimensional vector $\mathbf{x}_i \in \Re^d, i = 1, \ldots, l$ and the associated "truth" $y_i \in \{-1, 1\}$ provided by a trusted source. Here, $y_i = 1$ labels a positive example—in our application, the observed success of a controller applied when the dynamic figure is in state $\mathbf{x}_i$—while $y_i = -1$ labels a negative example—the failure of the controller applied to state $\mathbf{x}_i$. The set of observations $\{\mathbf{x}_i, y_i\}$ is called the *training set*. The SVM is a machine whose task is to learn the mapping $\mathbf{x}_i \mapsto y_i$ from a training set. The SVM is defined by functional mappings of the form $\mathbf{x} \mapsto f(\mathbf{x}, \alpha)$, where α are parameters. A particular choice of α

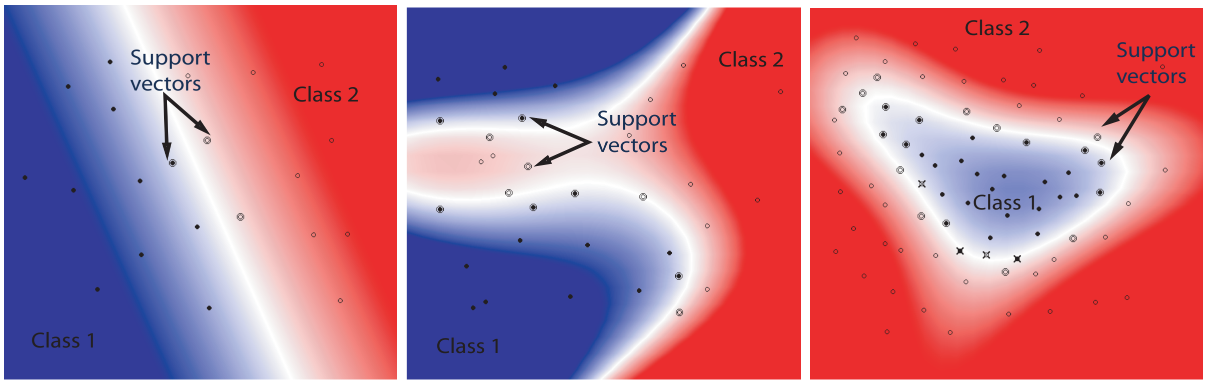


Figure 4.2: Two dimensional SVM classifier.

generates a "trained" SVM. In a trained SVM, the sign of the *decision function* $f(\mathbf{x})$ represents the class assigned to a test data point $\mathbf{x}$. In our application, a properly trained SVM predicts if a controller will succeed ($f(\mathbf{x}) > 0$) or fail ($f(\mathbf{x}) < 0$) on a given state $\mathbf{x}$ of the dynamic character.

How does one train an SVM? In the simplest case of a linear SVM with separable training data, there exists a *decision boundary* separating positive from negative examples which takes the form of a "separating hyperplane" in $\Re^d$. The SVM training algorithm computes the separating hyperplane with the largest *margin* $d_+ + d_-$, where d_+ (d_-) is the shortest distance from the separating hyperplane to the closest positive (negative) example. SVM training requires the solution of a quadratic programming optimization problem involving a Lagrange multiplier α_i for every data point in the training set. Those data points in the solution with corresponding $\alpha_i > 0$ are called *support vectors*.

The support vectors are critical elements of the training set. They lie closest to the separating hyperplane. If other observations in the training set are moved (subject to certain restrictions) or removed and SVM training is repeated, the same separating hyperplane will result. To use a trained SVM, we simply determine on which side of the decision boundary a given test data point $\mathbf{x}$ lies and assign the corresponding class label to that point. The linear SVM is easily generalized to nonseparable training data.

Furthermore, it is straightforward to generalize the theory to encompass nonlinear SVMs for which the decision boundaries are no longer hyperplanes (i.e., the decision function are no longer linear functions of the data). The trick, in principle, is to map the data to some higher (possibly infinite) dimensional space in which the linear theory can be applied. This is easily done by introducing *kernel functions* $K(\mathbf{x_i}, \mathbf{x_j})$, such as the polynomial kernel $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$, or the Gaussian or radial basis function (RBF) kernel $K(\mathbf{x}, \mathbf{y}) = \exp(-|\mathbf{x} - \mathbf{y}|^2/2\sigma^2)$. For the mathematical details, we refer the reader to [14].

Figure 4.2 shows a 2-dimensional case involving two classes separated using three kinds of kernels. The linear kernel produces a linear decision boundary (left), the polynomial kernel produces a non-linear one (center) while the radial kernel produces an enclosing decision boundary (right). The support vectors are circled. These images have been produced by the SVT applet from Bell-Labs [10].

4.5 Applying SVMs

To apply the SVM technique to the problem of determining controller pre-conditions, we train a nonlinear SVM classifier to predict the success or failure of a controller for an arbitrary starting state. Thus, the trained SVM demarcates the boundary of regions in the figure's state space wherein the controller can successfully do its job. Training sets comprising examples $\{\mathbf{x}_i, y_i\}$ are generated by repeatedly starting the dynamic figure at a stochastically generated initial state $\mathbf{x}_i$, numerically simulating the dynamics of the figure under the influence of the controller in question, and setting $y_i = +1$ if the controller succeeds or $y_i = -1$ if it fails.

The distribution of the stochastically generated initial states is important. The sample points should ideally be located close to the boundaries which demarcate the acceptable pre-condition region of state-space. However, these boundaries are in fact the unknowns we wish to determine and thus we must resort to a more uniform sampling strategy. Unfortunately, the high dimensionality of the state-space precludes regular sampling. We thus adopt the following stochastic process to generate a suitable distribution of initial states: First, a nominal initial state is chosen, based upon the designer's knowledge of the controller. A short-duration simulation (typically 0.3s) is then carried out from this initial state while a randomized perturbation process is executed. This currently consists of applying an external force of random (but bounded) magnitude and random direction to the center-of-mass of the pelvis. Simultaneously, the character's joints are perturbed in a stochastic fashion by setting randomized offset target angles for the joints and using the character's PD joint controllers to drive the joints towards these perturbed positions. The perturbations are in the range of [-1.0, 1.0) radians. The combined effect of the random force, the reaction forces from the ground and the momentum that the motion of the character induces, results in a wide range of perturbed states from both sides of the boundary. While the perturbation strategy is admittedly ad-hoc, we have found it to be effective in sampling the pre-condition space, as is validated by the online use of the learned pre-condition models. Section 8.3 discusses ways of producing the training set that we might explore in the future.

We employ T. Joachims' SVM^{light} software which is available on the WWW [57]. The software can accommodate large training sets comprising tens of thousands of observations and it efficiently handles many thousands of support vectors. It includes standard kernel functions and permits the definition of new ones. It incorporates a fast training algorithm which proceeds by solving a sequence of optimization problems lower-bounding the solution using a form of local search. It includes two efficient estimation methods for error rate and precision/recall. The software provides various parameters that the user can adjust to control the behavior of the classifier. For example, a command line option allows the user to define the cost-factor by which training errors on positive examples outweigh errors on negative examples. Appendix A shows the complete list of parameters that SVM^{light} supports and their default values. For our experiments we use the default values for all parameters, except for the kernel one for which we use option "1" (polynomial).

4.6 Results

The SVM training phase can take hours in our application, but this is done off-line. For example, on a 733 MHz PIII computer, the SVM training time for a training set of 8,013 observations is 2,789 seconds using the polynomial kernel, 2,109 seconds using the linear kernel, and 211 seconds using the radial kernel. For a training set of 11,020 observations, the training time

Controller	Size of train set	#positive	Size of test set	#positive
DStanceToCrouch	8999	3155	9110	3260
ProneToKneel	4200	3965	4223	4008
SupineToKneel	2234	2234	1879	1879
CrouchToStand	6926	3707	14272	7594
Balance	17317	6997	20393	2683
Walk	11020	1760	8658	1453
StandToSit	1100	458	1286	559
StandToStep	16999	4445	17870	4570
KneelToStand	6000	1063	11998	2806

Table 4.1: Training and test set sizes.

Controller	SVM(+)	SVM (-)	Total	NN (+)	NN (-)	Total
DStanceToStand	84	89	87	72	86	81
ProneToKneel	100	51	94	97	26	93
SupineToKneel	100	-	100	100	-	100
CrouchToStand	100	100	100	98	98	98
Balance	63	91	88	42	90	84
Walk	91	98	98	78	96	93
StandToSit	63	75	70	59	68	64
ProtectStep	48	90	79	49	81	72
KneelToStand	74	88	85	65	84	79

Table 4.2: Comparison between learned SVM and NN pre-conditions.

is 8,676 seconds using the polynomial kernel, 3,593 seconds using the linear kernel, and 486 seconds using the radial kernel. Table 4.1 shows the size of the training and test sets that we have used, and indicates the number of successful (positive) examples in each set. It is worth noting that the SVM code we use works iteratively. The training time is not constant with respect to the size of the training set. It depends mainly on the shape of the boundary and to our knowledge convergence is not guaranteed. Once trained, the SVM classifier can provide answers on-line in milliseconds which is essential for interactive applications. An interesting feature of the method is that the classification algorithm produces an approximation of the boundary based only on the support vectors. The number of support vectors in most of our experiments is much smaller than the number of training data points. For example, a training set of size 10,000 can be reduced to 700 support vectors. Part of the efficiency of the classifier stems from this reduction.

Through systematic experimentation, we have evaluated the performance of our automatic, SVM-based algorithm for learning controller pre-conditions. We compared the performance of the SVM algorithm to that of a *nearest neighbor* (NN) classifier, Duda [26]. Given a training set, the nearest neighbor classifier returns, for an arbitrary state $\mathbf{x}$, the same succeed/fail label as the label of the observation in the training set that is closest to $\mathbf{x}$. The distance between observations is computed using a Euclidean norm, $dist = |\mathbf{x} - \mathbf{x}_i|$. All observations are in *rads* or *rads/sec*. NN classifiers should perform particularly well in cases where the feasible area in the state space is highly fragmented and localized. Note that the NN method requires zero training time, but that it provides an answer in $O(n)$ time where n is size of the training set. However, this response time can be improved by building appropriate data structures.

Table 4.2 summarizes the percentage success rates of learned pre-conditions for a variety of controllers that we use later in our demonstrations. The "(+)" indicates results that correspond to test sets that consist exclusively of positive (successful) examples, while the "(-)" sign corresponds to results for test sets that consist of negative examples. To compute accuracy rates, we trained the SVM and NN pre-condition learning algorithms using randomly sampled observations collected from each of the controllers. Then we generated test sets of novel observations and compared their true success/fail status against that predicted by the trained NN and SVM pre-conditions to obtain the accuracy percentages listed in table. The results show that the SVM algorithm consistently outperforms the NN classifier. For the results shown in the table, the SVM algorithm employed polynomial kernel functions. We ran a similar set of experiments using Gaussian RBF kernel functions, but the accuracies were consistently lower than those obtained with polynomial kernel functions. Note that for the supine-to-kneel controller the training set contains no negative results. The process through which we produce the training sets, Section 4.5, failed to produce negative cases. This is due to the strong stability of the supine position.

Machine learning methods often have to use noisy example sets since for some applications the collection of the example data happens through noisy sensors. This would be the case if we were using our method on a robot. In our case, noise is not an issue since we deal with simulated data which is free of noise. However, it is informative to know how our method would work in the presence of noise as well as irrelevant sensory data. In a first experiment, we increase the size of the character's state by one element both in the training and the test sets. The values of this element come from randomly sampled white noise of magnitude 1.5 *rads*. The results for three controllers are shown in Table 4.3. Columns three and four show the number of support vectors in the model for the clean and the noisy case respectively. It is clear that although the added noisy element has some effect, the effect is relatively small. They are an indication that SVM-based methods are able to ignore an irrelevant element in the state of the character.

Controller	Succes no noise %	#Sup. vecs	Success with noise %	#Sup. vecs
DStanceToStand	87.29	1604	86.68	1660
Walk	96.73	738	96.17	862
CrouchToStand	99.77	41	99.73	54

Table 4.3: SVMs ignore an additional noisy element.

Our second experiment involves adding noise of different magnitudes to every element in the training set. Table 4.4 shows the percentages of success and the number of the support vectors for three controllers. Note that the performance of the *double stance-to-crouch* controller increased after adding noise of magnitude 0.01. A possible explanation for this might be that the noise reduces a mild overfitting effect (see Section 4.1) and increases the generalization ability of the classifier.

It is interesting to note the correlation between the support vector machine and the NN results. Although, we do not have a rigorous theoretical explanation for that, we think it is due to the fact that both methods work in a similar way. They model clusters using a polynomial scheme. The larger these clusters are, the fewer opportunities for misclassification. Most classifications methods favor domains that consist of smooth clusters because the simpler the class boundaries are, the better they can be modeled by the hypothesis space of the classifier.

The next chapter discusses the controllers we have developed, and their composable API. Section 5.6, in particular, describes the way we combine the manual method and the SVM-classifier to improve the modeling of the pre-condition region.

Noise	Success (SVM) %	# Sup. vecs	Success (NN)
0.00	87.29	1604	80.97
0.01	87.48	1625	81.05
0.10	87.23	1811	80.87
1.00	81.86	3719	65.70

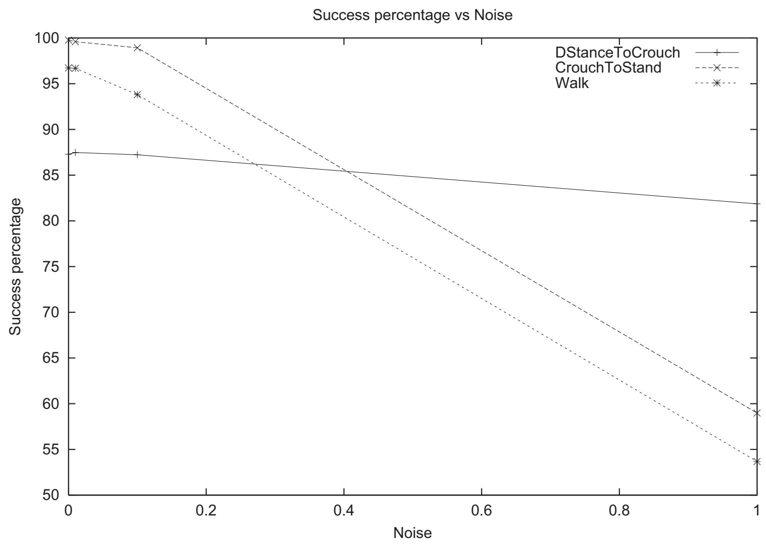
(a) DStanceToCrouch

Noise	Success (SVM) %	# Sup. vecs	Success (NN)
0.00	99.77	41	98.05
0.01	99.59	66	97.91
0.10	98.83	394	95.79
1.00	58.98	3611	31.11

(b) CrouchToStand

Noise	Success (SVM) %	# Sup. vecs	Success (NN)
0.00	96.73	738	92.78
0.01	96.69	826	92.68
0.10	93.81	1717	89.85
1.00	53.68	4073	67.61

(c) Walk



(d) SVM success percentage vs noise.

Table 4.4: SVMs performance at the presence of noise.

Chapter 5

Simulation

Human and animal characters are widely used in animation applications. Video games, science fiction movies, animated movies such as PIXAR's *Toy Story* all feature a variety of animated humanoid characters. Such characters are modeled using kinematic or dynamic *articulated figures*. We begin this chapter with background information about physics-based animation techniques and control of animated characters. We then describe the specific composable controllers that we have developed to test our composition framework.

5.1 Physics-based simulation of articulated figures

An *articulated figure* is a collection of rigid links connected with joints. Usually there is a clear hierarchy between the links, each link is connected to one parent link while a parent link may have more than one child links. Each pair of parent-child links connects through a unique joint that defines the motion of the child link with respect to its parent. In most cases the joints are purely rotational allowing a maximum of three rotational degrees of freedom for each link. The highest link in the hierarchy is called the *root* link and has a total of six degrees of freedom that corresponds to its position and orientation in space. Figure 5.1 shows an articulated figure that represents a human skeleton. Because of the clear hierarchy and relation between the links it is often convenient to describe the state of an articulated character using a reduced coordinate system that is defined by the position of the root link and the orientation of each link with respect to its parent (see Featherstone [29] and SD/Fast [49]). Alternatively we could use a Cartesian coordinate system that considers the world position and orientation of each link separately (see Baraff [6] and Mathengine [70]). However, the reduced coordinate system approach is more convenient for simulation and for implementing control algorithms. It is the therefore the approach we use in this thesis. The collection of all degrees of freedom, $\mathbf{q}$ of the character coupled with their velocities $\dot{\mathbf{q}}$ represent the *state* of the character. The state of the character as a function of time uniquely describes the motion of the character.

The motion of an articulated figure is described by a set of second order differential equations. These equations result from the application of Newton's law and relate the acceleration of the degrees of freedom, $\ddot{\mathbf{q}}$, of the system with the forces that act on it. Systems that model active characters, such as humans, are subject to both external forces that come from their environment and internal muscle forces.

The equations of motion of an articulated figure have the general form:

$$\mathbf{M}(\mathbf{q}(t))\ddot{\mathbf{q}}(t) + C(\mathbf{q}(t), \dot{\mathbf{q}}(t)) = \sum \mathbf{J}_T^T \mathbf{F}_i + \sum_l \mathbf{J}_R^T \tau_{ext,l} + \sum_k \mathbf{J}_R^T \tau_{j,k}, \tag{5.1}$$

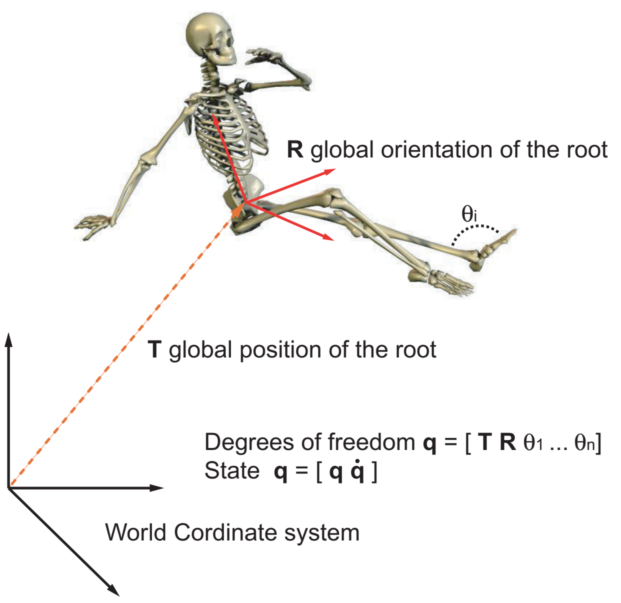

R global orientation of the root
θi
T global position of the root
Degrees of freedom **q** = [**T** **R** θ1 ... θn]
State **q** = [**q** **q̇**]
World Cordinate system

Figure 5.1: An articulated character.

where $\mathbf{q}$ represents the vector of reduced coordinates, $\mathbf{M}$ is a symmetric, positive definite mass matrix, C are gyroscopic forces, $\mathbf{J}$ is the Jacobian matrix, $\mathbf{F}$ are the external forces, τ_{ext} the external torques and, in the case of an active system, τ_j are the torques exerted by the system's actuators (muscles). Featherstone [29] and Marion [69] present a detailed description of the dynamics of systems such as articulated bodies. Equation 5.1 represents a system of non-linear second order differential equations that has no analytical solution for complex articulated characters. For such cases the system is discretized with respect to time and solved numerically.

5.1.1 Numerical solution of the equations of motion

The standard way of solving a complex dynamical system such as the one described by Equation 5.1 is to consider the motion of the system in a series of small time intervals. Within each time interval the vector of accelerations $\ddot{\mathbf{q}}$ is computed as many times as the integration method requires. The accelerations can be computed by transforming Equation 5.1 into a linear system $\mathbf{M}\ddot{\mathbf{q}} = \mathbf{b}$ and solving for the accelerations. The total applied force $\mathbf{b}$ is computed given a model of the environment and the internal actuation of the character. Integrating the computed accelerations twice yields the positions and velocities of the degrees of freedom at the end of each time interval.

The size of the time interval, typically called time step, has a direct effect on the accuracy of the simulation; a smaller time step provides a better approximation of the continuous motion of the system and hence more accurate results. There is a clear trade off between accuracy and efficiency. Often a relatively large time step may cause instability and oscillations. In certain cases, the choice of integration method can allow for larger time steps and increase the efficiency of the system. Lately, implicit and semi-implicit integration methods have been used in a variety of applications [99, 9] to allow for larger time steps. However, they cannot be easily applied in cases where the external forces have an impulsive nature, for example in the case of collision reaction forces.

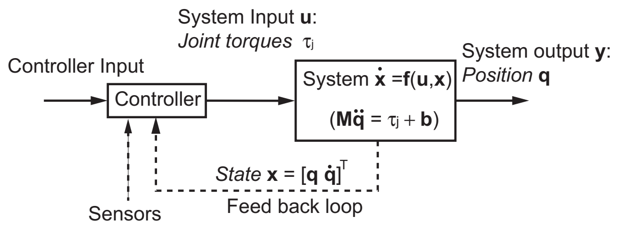

System Input **u**:
Joint torques τ_j
Controller Input
Controller
System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{u},\mathbf{x})$
$(\mathbf{M}\ddot{\mathbf{q}} = \tau_j + \mathbf{b})$
System output **y**:
Position **q**
State $\mathbf{x} = [\mathbf{q}\ \dot{\mathbf{q}}]^T$
Sensors
Feed back loop

Figure 5.2: Controlling an articulated character.

5.2 Control

Kinematic techniques allow the user to directly specify the value of the character's degrees of freedom over time. Therefore the concept of animating a character is reduced to producing the motion curves that the degrees of freedom of the character will follow precisely. Physics-based techniques simulate the real world where the motion of an object is determined as the result of applied forces. In this case, animating a simulated character requires computing the muscle torques that the character must exert at its joints in order to perform a desired motor task. This is a difficult control problem. To deal with this problem we borrow ideas and notation from control theory, Dorf [25]. Given a dynamic system with a set of input parameters $\mathbf{u}$, a set of output parameters $\mathbf{y}$, a set of state parameters $\mathbf{x}$ and initial values $\mathbf{x}_0$ for the state parameters, a *controller* is an algorithm or function that produces the input parameters that will drive the output parameters to desired values. A standard way to mathematically represent the control of a dynamic system is as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{5.2}$$
$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)), \tag{5.3}$$
$$\mathbf{x}(t_0) = \mathbf{x}_0. \tag{5.4}$$

where $\mathbf{f}$ is the mathematical model of the system and $\mathbf{g}$ is the control function, i.e the function that computes the output from the state and the input parameters. Generally, it is not easy to invert $\mathbf{g}$. In our case, the system under control is an animated character described by Equation 5.1, the input parameters are the muscle torques $\tau_{\mathbf{j}}$, the output parameters describe the position of the character i.e. its degrees of freedom $\mathbf{y} = \mathbf{q}$, and the state of the character is $\mathbf{x} = [\mathbf{q}\ \ \dot{\mathbf{q}}]^T$.

Control techniques that do not require feedback on the state of the system are called *open-loop* while those that monitor the state of the system are called *closed-loop*. Figure 5.2 shows a graphical representation of a system and a controller. The dotted line shows the state feedback loop that is used in the closed-loop case. It is often the case that controllers cannot directly observe the state of the system, in which case they estimate it from sensory input. In addition, sensors can provide additional information about the environment and the system under control, such as contact points, the location of obstacles and the orientation and slope of the terrain.

5.3 Design methods

Determining motor control strategies for human motor tasks and describing them in an algorithmic way so that a simulated character can reproduce them, is surprisingly difficult even for everyday motions such as walking. Wooten's thesis [113] is a good example of how much effort it takes to design robust controllers for athletic maneuvres.

The biomechanics literature is a good source of information about the motor control of the human body in any level of detail. However, the focus of biomechanics is understanding and analyzing the human motion for purposes of increasing athletic performance, avoiding injuries, rehabilitation etc. Although, the associated studies rarely reveal how such analyses can be transformed into robust controllers they provide information that can be used in the design of control strategies for simulated characters. We will see how such information has been used in subsequent sections of this chapter.

In addition to the biomechanics literature, our own intuition is a good starting point towards designing controllers for simulated characters. After all, we can personally perform most of the motor tasks that we wish the simulated characters to reproduce. A number of the controllers we present in the next sections were developed based on observations of human subjects performing the tasks repeatedly, such as illustrated in Figure 5.11.

The simplest and most common method for developing controllers involves much *trial and error*. Using this technique, a first version of a controller is developed, usually by hand. The controller is then repeatedly tested and manually refined until its performance is satisfactory. The control problem can often be formulated as an *optimization problem*. The character is required to perform a task while minimizing a cost or maximizing an objective function. It has been shown that optimization-based control can be used to automatically synthesize controllers capable of making active simulated creatures locomote [39, 96, 41, 100]. Controller-based motion synthesis is often addressed using probabilistic optimization methods because they are easy to implement, are suitable for searching large spaces, and can avoid local sub-optima. With this method, controllers are repeatedly generated and subsequently evaluated using forward simulation. The motion synthesis problem is thus tackled by searching the space of possible controllers for those that produce suitable motions. Common search methods for the stochastic motion synthesis problem are *genetic algorithms* [96, 39], and *simulated annealing* [58, 100, 27, 41]. The techniques based on probabilistic search work well for developing relatively stable motions such as crawling gaits. However, they have not been shown to work for highly unstable motions, such as bipedal walking.

5.4 Our control structures

Many different methods have been developed to deal with the control of dynamic graphics objects [115, 45, 72, 100, 41, 99, 62]. Finite state machines (FSM) are one of the most popular structures for the control of gait. The *pose controller* is a type of finite state machine which is suitable for the control of articulated figures [102]. The technique was originally based on cyclic graphs but it has been extended to acyclic graphs in order to achieve non-periodic motions [101]. Most of our controllers follow the *pose controller* paradigm as explained in Section 5.5.

A *pose controller* is a finite state machine (FSM) with timed and event-based transitions. Each state of the FSM corresponds to a particular configuration (pose) of the character and it remains active for a specified time interval or until a sensor triggers an event-based transition. For the rest of the document, we will use the term *pose* to refer to a *state* of the pose controller FSM since the term *state* also refers to the character's state $[\mathbf{q} \quad \dot{\mathbf{q}}]^T$. Each pose defines the desired value for the character's degrees of freedom $\mathbf{q}$. A set of *proportional derivative* controllers transforms these values into rotational spring forces as follows:

$$\tau = \mathbf{K}_s(\mathbf{q} - \mathbf{q}_{des}) - \mathbf{K}_d\dot{\mathbf{q}} \tag{5.5}$$

where $\mathbf{K}_s$ and $\mathbf{K}_d$ are diagonal stiffness and damping matrices respectively, $\mathbf{q}_{des}$ are the desired values of the character's degrees of freedom specified by the pose and $\dot{\mathbf{q}}$ is the velocity of the

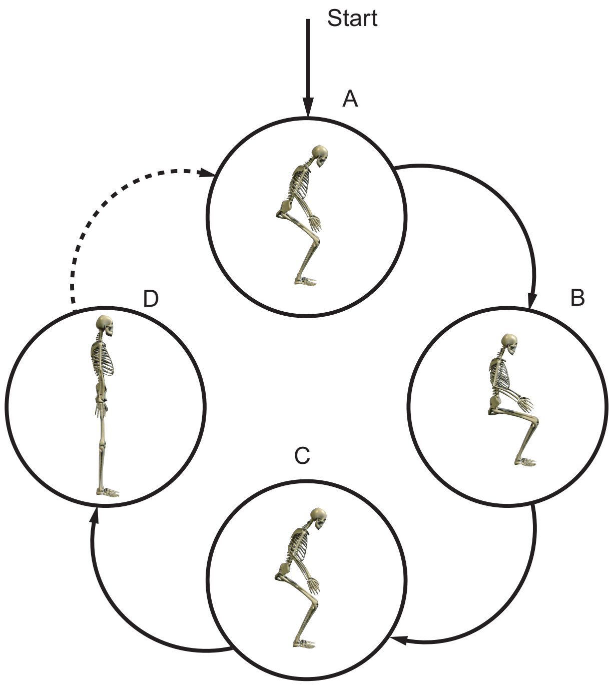


Figure 5.3: A stand-sit-stand pose controller.

character's degrees of freedom. Figure 5.3 shows the pose controller that implements a stand-sit-stand motion described in Section 5.8.3. It consists of four poses A,B,C,D. Transitions A $\rightarrow$ B and C $\rightarrow$ D are timed, while B $\rightarrow$ C is sensor based.

For some of our controllers, we use continuous control, meaning that the control parameters are tightly coupled with some of the feedback sensors. The balance controllers are such an example. We discuss sensors in Section 5.5.1.

We have designed several controllers based in part on experimental studies of how humans detect loss of balance [81] and analysis of protective and falling behaviors [24]. The resulting parameterized controllers have been enhanced with appropriate pre-conditions, post-conditions, and expected performance and have been integrated using our arbitration-based supervising controller.

With the exception of the supervisor controller, most of our controllers are designed specifically for the two models that we use in our experiments and they may not scale well for different models that vary in dimensions and weight. Similarly, most of them are sensitive to the ground model and may fail if a slightly different one is used. These are limitations of the controllers themselves and not of the composition method.

5.5 Supervisor controller

The supervisor controller implements the top level of the hierarchical composition scheme shown in Figure 3.4. At each time step it first checks whether it needs to initiate a bidding process; if the user-specified target state has changed or if there is no active controller other than a default one, a bidding process is executed. During this process all available controllers determine whether their pre-conditions are satisfied in which case they bid for control of the character. The supervisor controller selects the bidding controller that returns the highest priority and registers

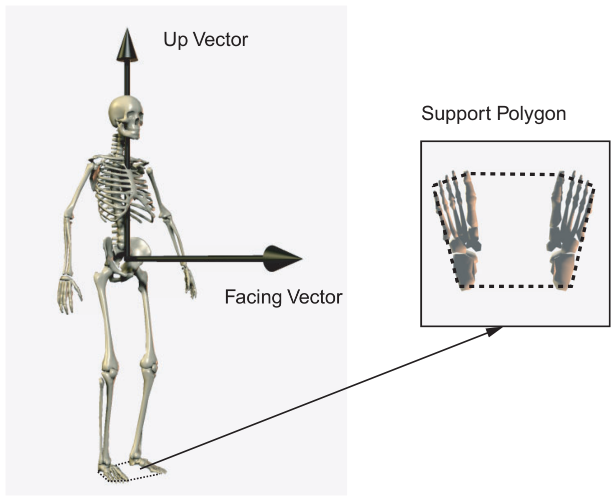


Figure 5.4: A few sensors associated with the 3D model.

it as the active controller. It then calls the method of the active controller that implements its control strategy. This method returns to the supervisor controller a status parameter and target values for all or a subset of the character's degrees of freedom along with associated stiffness and damping parameters. If the status parameter indicates that the controller has failed then a new bidding process is executed[1]. The target values and the stiffness and damping parameters are used by a set of proportional-derivative controllers to calculate the actual control torques. Alternatively, the active controller can choose to apply torques directly to the character and return no values for the supervisor's proportional-derivative controllers. In the case where no available controller bids for control, the supervisor controller activates the *default* controller. The latter is a generic controller that tries to do something sensible with the character when no specialist control can take over. We describe this controller in more detail later.

As the central control unit of a character, the supervisor controller is responsible for maintaining the joint limits of a character. We use a method based on *exponential springs* to ensure that rotations of the character's body parts do not exceed the user specified limits. If a rotational degree of freedom q_i, exceeds its allowable range of (q_i^{low}, q_i^{high}), the exponential springs produce the following forces:

$$\texttt{if} \quad (q_i^{low} - q_i) > \epsilon \rightarrow F_{low} = k_s^l (e^{k_s^e (q_i^{low} - q_i)} - 1) - k_d \dot{q}_i, \tag{5.6}$$

$$\texttt{if} \quad (q_i - q_i^{high}) > \epsilon \rightarrow F_{high} = k_s^l (e^{k_s^e (q_i - q_i^{high})} - 1) - k_d \dot{q}_i, \tag{5.7}$$

depending on the limit that has been violated. Exponential springs are widely used in a variety of control problems. In our current implementation we have found that the values, $k_s^l = 10.0, k_s^e = 1.0, k_d = 10.0$ of the exponential spring constants produce satisfactory behavior.

5.5.1 Sensors

Feedback is very important for the motor control of complex characters such as humans. Controllers need to know sufficient information about the environment such as contact points, the

[1]An additional check ensures that the system does not fall into an infinite loop when a badly designed controller bids for control and immediately fails.

slope of the terrain, the position of obstacles, etc. At the same time they need information on the state of the character, where he is facing, whether he is balanced, etc. Most of the information on the character can be computed from the state parameters. However it is often more convenient to use higher level sensors that are more intuitive and which can be computed once per time step and shared among controllers. In the current implementation, each controller has full access to the internal data structures of DANCE including all the information associated with any character or object in the system. This allows the controllers to define arbitrary sensors that keep track of necessary information such as state parameters for feedback loops and the state of the environment. For efficiency, the supervisor controller calculates a number of common sensor values that are available to all specialist controllers:

- *Support polygon.* The support polygon, $\mathcal{S}$, is defined by the convex hull of the feet and it is crucial for the balance of the character.
- *Center of mass information.* The position $\mathbf{c}$, velocity $\dot{\mathbf{c}}$, acceleration $\ddot{\mathbf{c}}$, and relative position of the center of mass with respect to the geometric center of the support polygon.
- *Hip center of mass information.* The position $\mathbf{c}^h$, velocity $\dot{\mathbf{c}}^h$, acceleration $\ddot{\mathbf{c}}^h$, and relative position of the hip's center of mass with respect to the geometric center of the support polygon.
- *Contact information.* Whether the feet, head, hip and thighs are in contact with the ground.
- *Orientation.* The *facing*, $\mathbf{v}^f$, and *up*, $\mathbf{v}^u$, vectors of the hip indicate the direction that the hip faces and how far it leans respectively.

Figure 5.4 shows the support polygon, the facing vector and the up vector relative to the 3D skeleton model.

5.5.2 Command interface

Some of our controllers bid for control of the character autonomously when their pre-conditions are met, such as the controller that takes a protective step in response to loss of balance. Other controllers attempt to become active only at the users request. Each controller is associated with a string that defines its goal. For example, the walk controller for the 2D robot model is associated with the string "walk *n*" where *n* is the number of steps that the character must take. Our system does not prevent controllers from using the same string if so desired. Such conflicts are resolved by the priority scheme described in Section 3.7. The user registers one goal at a time by interactively entering a command string to the supervisor controller. The new goal increases the suitability score of the designated controller and forces the supervisor controller to run the selection process that finds the most suitable controller. The selection and arbitration process could be implemented by a high-level planner. Section 8.1 discusses how we might investigate this direction in the future.

5.6 Implementing the composable API

Chapters 3 and 4 present two techniques that can be used to model the pre-conditions of composable controllers, one being a manual specification and the other using a semi-automated

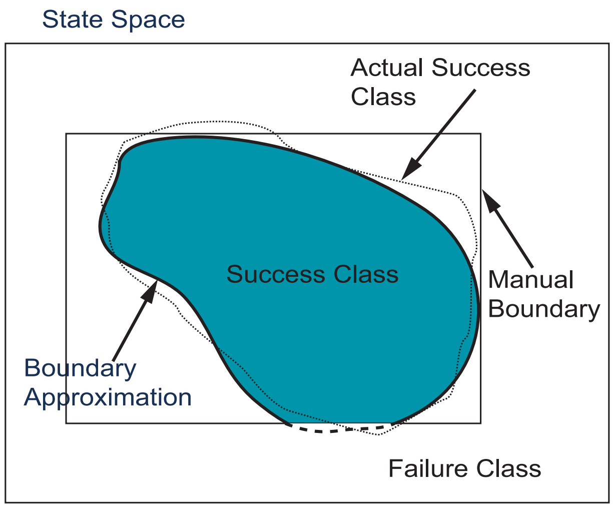


Figure 5.5: Manual and learned approximations of the success region.

application of machine learning. The technique based on manually defined analytical pre-conditions is convenient when the operation of the controller depends on conditions that can be represented in closed form, such as the position of the center of mass with respect to the support polygon, contact points. These conditions define regions with regular shapes within some projection of the state space. On the other hand, the classifier based on machine learning provides a more flexible and semi-automatic way of computing the region of success within the full state space of the character. However, if we do not add explicit information about the environment to the training set of the classifier, the latter will fail to capture important environmental conditions. Therefore, it seems that the best way to model the pre-conditions of a composable controller is to combine the two techniques in a way that leverages the advantages of both. We have decided to use the manual pre-conditions first to prune out large parts of the state space and ensure that contact and balance conditions are properly captured. Within the remaining region we apply a support vector machine classifier that results in a more accurate prediction. Figure 5.5 shows an abstraction of how the different regions might look like in the state space. The square line represents the approximation obtained by the manual approach, the boundary produced by the classifier is shown as a solid line and the dotted line represents the actual region of success.

We use the learning technique only for the pre-conditions of the controllers associated with the two dimensional robot model. The use of the support vector machine classifier for the three dimensional case is straightforward but more computationally expensive. This is mostly due to the much greater dimensionality of the three dimensional case compared to the two dimensional one. In particular, the former is 2.65 times larger than the latter. A much larger state vector has a significant effect on the efficiency of the dynamics simulation which is needed to produce the training set for the learning process. The 3D dynamics simulation runs, on average, 3-7 times slower than the 2D one.

The next sections present the controllers we have implemented and describe in detail their analytical composable APIs. Table 5.1 shows the number of poses (states) of each controller. Controllers that do not follow the *pose controller* (finite state machine) paradigm are marked with “C”. The table shows also whether the controllers implement sensor-based (“S”) or time-based (“T”) transitions between poses.

Controller	#poses (3D)	Transitions	# poses (2D)	Transitions
Balance	C	N/A	C	N/A
Fall	3	T & S	3	T & S
ProtectiveStep	4	T & S	5	T & S
Step	9	T & S	-	-
Plunge	3	T & S	6	T & S
SupineToCrouch	10	T & S	-	-
SupineToKneel	-	-	10	T & S
ProneToCrouch	12	T & S	-	-
ProneToKneel	-	-	6	T & S
RollOver	11	T & S	-	-
CrouchToStand	C	N/A	C	N/A
Kip	9	T & S	N/A	N/A
StandToSit	3	T	3	T & S
SitToCrouch	1	T & S	1	T & S
StandToAllFour	C	N/A	-	-
DStanceToCrouch	-	-	6	T & S
Slow Steps (Walk)	-	-	6	T & S

Table 5.1: Number of poses and pose transition types.

5.7 Default Controller

The default controller is activated when no other controller requests control of the character. Its goal is to perform a sensible action in any given situation. In the absence of a better understanding of the situation, the most sensible thing to do is to keep the character in some comfortable position. We currently distinguish between two different situations, standing in place and lying on the ground. In the first case, the controller attempts to maintain the character's upright stance using moderate force while keeping the arms loose. If the character is leaning more than a given threshold then it is considered to be in a lying position, in which case the controller makes the character assume a relaxed pose. Thus far, these two strategies have worked well, in the sense that they bring the character smoothly into a perceived comfortable position.

The default controller faces the difficult task of encompassing all situations for which we haven't yet designed appropriate controllers. It is therefore the starting point of future improvements.

5.8 Everyday Actions

A skillful simulated human character should be able to perform all the motor tasks that humans are able to do. However, even very common tasks such as walking require the sophisticated control of body dynamics. As stated earlier, we focus on a subset of everyday motions starting with the most simple one–standing in place. In the event of a loss of balance, the character should react naturally either with a restorative motion or with a protective falling behavior, as is appropriate in the specific circumstance. Affording a dynamic articulated figure natural reactions to a loss of balance or an impending fall, plus the ability to rise up subsequent to a fall, is an essential step towards believable, autonomous characters.

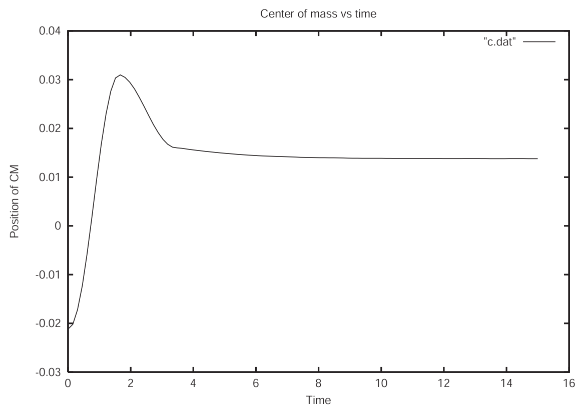


Figure 5.6: Critically damped balance controller.

5.8.1 Balancing

Balancing during quiescent stance is a complex biomechanical control phenomenon that depends on different factors, such as the distance between the feet, and the presence of (or lack of) visual feedback, Day [23]. A considerable body of research aims to understand the sensory information (van der Kooij [106]), and reflex responses that humans use to maintain quiet stance (Fitzpatrick [32]). The strategies that people employ as a response to disturbances during quiet stance are generally divided into *hip strategies* and *ankle strategies* depending on whether the hips or the ankles are the dominant regulators of the postural stability. Gatev [34] provides a comprehensive analysis of balance strategies during quiet stance focusing on ankle control. Most researchers in biomechanics seem to agree that ankle strategies are more likely to occur in response to small disturbances, while hip strategies occur in response to larger disturbances.

Our *balance* controller is responsible for maintaining a natural standing posture. It is based on an inverted pendulum model that uses the ankles to regulate the body sway [32]. Despite the fact that the body of the character is not as rigid as the inverted pendulum hypothesis suggests, the approximation works well in practice. Our balance controller uses an ankle value of 0.06 radians as the equilibrium position. This value has been chosen experimentally. Figure 5.6 shows the horizontal trajectory of the center of mass in the sagital plane that the balance controller produces when the character's starting state places the center of mass close to the heels. The critically damped behavior of the balance controller is apparent.

For this controller, the articulated body must be in a balanced upright position, the velocity and acceleration of the center of mass should not exceed certain threshold values as explained by Pai [81], and both feet must maintain contact with the ground at all times. The controller can tolerate small perturbations of the posture and the velocity/acceleration of the center of mass by stiffening the ankle joints. For larger accelerations of the center of mass, the controller actively actuates the ankle joint to reduce the acceleration of the center of mass. The post-conditions are similar to the pre-conditions. The expected performance is exactly the same as the pre-conditions. In mathematical form using the notation defined in Section 3:

$\mathcal{P}$:

Velocity: $|\dot{\mathbf{c}}| < 0.3$ m/sec.

Balance: projection$(\mathbf{c}) \in \mathcal{S}$.

Posture: (upright) $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.1$ rad,

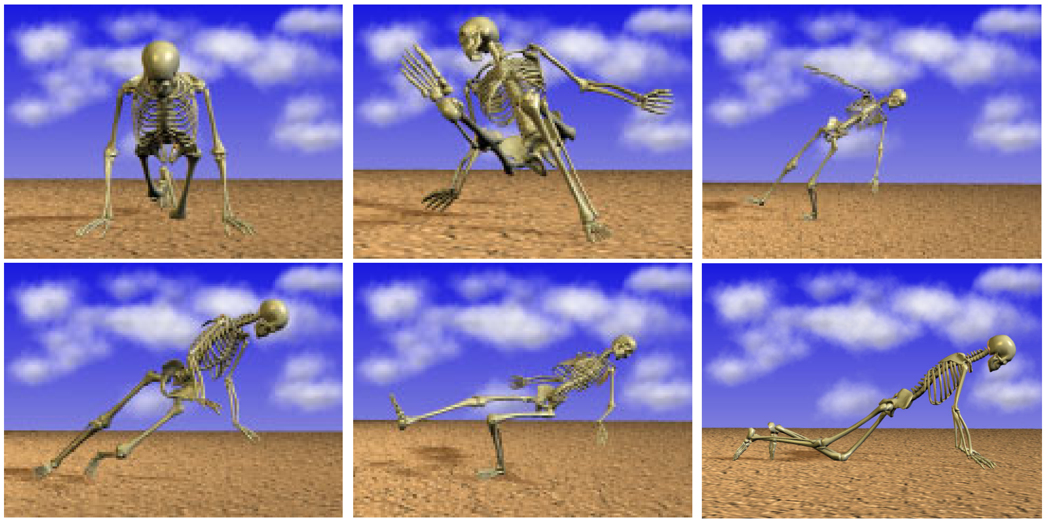


Figure 5.7: Falling in different directions

where $i = (\text{thigh}, \text{knee}, \text{waist})$, $\mathbf{q}_0 = \mathbf{0}$,
and n is a normalization parameter.

Contact: feet on ground.

$\mathcal{O}$:

Velocity: $|\dot{\mathbf{c}}| < 0.05$ m/sec.

Balance: projection($\mathbf{c}$) $\in \mathcal{S}$.

Posture: (upright) $(1/n) \sum_i \sqrt{(q_i - q_{0,i})^2} < 0.1$ rad,
where $i = (\text{thigh}, \text{knee}, \text{waist})$, $\mathbf{q}_0 = \mathbf{0}$,
and n is a normalization parameter.

Contact: feet on ground.

We enhance the behavior of our balance controller in a simple fashion by kinematically simulating the character's visual attention. In particular, we apply Perlin noise, Perlin [86], to the degrees of freedom of the neck that makes the character look around in its environment.

Because of the relative simple task that this controller has to accomplish and the inherent stability of the simple ankle strategy that we employ, the balance controller can be used successfully on slightly different terrains and characters. Nevertheless, the controller could be enhanced to employ more complex strategies especially as responses to larger external disturbances. For example, an animated character should attempt to maintain balance by shifting its weight, or bending at the waist. If the character cannot maintain balance, it must then resort to taking a step or even initiate a fall behavior.

Wooten's balance controller [113] implements a similar balance technique based on both hip and ankle strategies.

5.8.2 Falling

The manner in which people fall depends upon a number of factors such as their physique, their age and their training. Involuntary falling reactions are very common in everyday life, especially among young children and the elderly. They are probably the most common reason behind fracture injuries among the elderly. Hsiao and Robinovitch [50] show that, during a fall, the elderly are more likely to impact their hip first as compared to younger adults falling under the same conditions. Our *fall* controller is designed with the average adult in mind. Its main action is to absorb the shock of the impact using mostly the hands.

Wu [114] provides a way to distinguish falls from normal activities based solely on velocity characteristics. The pre-conditions of our fall controller define a larger acceptable region in velocity space than the one specified by Wu [114] because they are defined in accordance with those of the balance controller. All situations that are beyond the capabilities of the latter should be handled by the fall controller:

$\mathcal{P}$:
Vertical Velocity: $\dot{c}_v < 0.3$ m/sec.
Balance: projection($\mathbf{c}$) $\notin \mathcal{S}$.
Contact: hip not on ground, hands not on ground.
$\mathcal{E}$:
If falling forward face down $v_y^f < 0.1$.
If falling backward face up $v_y^f > -0.1$.
Contact with the ground in 3 seconds.
$\mathcal{O}$:
Either
Velocity: $|\dot{\mathbf{c}}| < 0.3$ m/sec.
or
head on ground.

The pre-conditions ensure that if the character is not balanced then the fall controller bids to take over. The fall controller succeeds when the velocity and acceleration of the character are brought close to zero or when the head touches the ground. The expected performance ensures that the character keeps on falling towards the same direction. In addition, it requires (a) that the character's facing direction does not reverse, something which might happen when falling from a great height and (b) that the character touches the ground within 3 seconds to ensure that it is a fall from a short height.

Our implementation of the fall controller computes the direction of the fall and responds accordingly. It can therefore handle a variety of pushes. Figure 5.7 shows snapshots of falls in different directions. The second frame in Figure 1.3 also demonstrates the action of the fall controller within a fall-and-recover sequence. The controller is relatively robust and it can be used on different characters and ground models.

5.8.3 Stand-to-sit and sit-to-crouch

Sitting down on and rising from a chair are common actions. We have implemented a controller that makes the character sit starting from an upright stance and another controller that prepares the character for the reverse action by making it lean forward until he is in a crouch position.[2] The resulting actions are illustrated in Figure 5.8. The pre-conditions, post-conditions and expected performance of each controller are relatively simple:

Stand to sit controller:
$\mathcal{P}$:
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.

[2]We use the term *crouch* to refer to any balanced posture of the character for which the legs of the character are symmetrically positioned.

Posture: (upright) $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.1$ rad,
where $i = (\text{thigh}, \text{knee}, \text{waist})$, $\mathbf{q}_0 = \mathbf{0}$,
and n is a normalization parameter.
Balance: projection($\mathbf{c}$) $\in \mathcal{S}$.
Contact: hip not on ground, hands not on ground.

$\mathcal{E}$:
Up vector: $v^u_y > 0.7$.
Requires that the character does not lean sideways, $|v^u_z| < 0.05$.

$\mathcal{O}$:
Up vector: $v^u_y > 0.7$.
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.
Requires that the character does not lean sideways, $|v^u_z| < 0.05$.

Sit to crouch controller:

$\mathcal{P}$:
Up vector: $v^u_y > 0.7$.
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.
Posture: sitting: $(1/n)\sum_i |\mathbf{q}[i] - \mathbf{q}_0| < 0.5$ rad,
where $i = (\text{thigh}, \text{knee}, \text{waist})$,
$\mathbf{q}_0 = [-1.5 \;\; 1.5 \;\; 0.0]$,
and n is a normalization parameter.
Balance: projection($\mathbf{c}$) $\notin \mathcal{S}$.
Contact: hip not on ground, hands not on ground.

$\mathcal{E}$:
Up vector: $v^u_y > 0.7$.
The character does not lean sideways, $|v^u_z| < 0.05$.
Time of completion less than 4 seconds.

$\mathcal{O}$:
Up vector: $v^u_y > 0.7$.
The character does not lean sideways, $|v^u_z| < 0.05$.
Balance: projection($\mathbf{c}$) $\in \mathcal{S}_s$.

Here, $\mathcal{S}_s$ is subset of $\mathcal{S}$ shorter along the front-to-back axis, so as to ensure a more balanced final posture. If the controller does not succeed in four seconds then something is assumed to be wrong and the controller aborts. We have tested this controllers with chairs of height $40cm$.

5.8.4 Rising from a supine position

Rising off the ground is a surprisingly difficult motion to simulate. It involves rapid changes of the contact points and significant shifting of the character's weight. In addition, the frictional properties of the ground model greatly influence on the motion.

For the three dimensional model, the pre-conditions require that the character be lying with his back flat on the ground, within some tolerance. The post-conditions require that the character be balanced on its feet, with the feet side by side but not necessarily straightened up. The expected performance makes sure that the character does not fall sideways and that it completes its task within 20 seconds.

$\mathcal{P}$:
Facing vector: $v^f_y > 0.97$

Velocity: $|\dot{\mathbf{c}}| < 0.005$ m/sec.
Contact: hip on ground.
The character does not lean sideways, $|v_z^u| < 0.05$.

$\mathcal{E}$:
The character does not lean sideways, $|v_z^u| < 0.05$.
Facing vector: $v_y^f > -0.2$.
Up vector: $v_y^u < 0.99$ if the hip is not on the ground.

$\mathcal{O}$:
Balance: projection($\mathbf{c}$) $\in \mathcal{S}$.
Contact: hip not on ground.

A snapshot of a resulting motion is shown in Figure 5.9. This controller is very sensitive to the character and the ground's friction model, which has a coefficient of friction of 0.6.

For the two dimensional model, the pre-conditions are the same as for the three dimensional case. However, the expected performance and the post-conditions require that the character ends up in a kneeling position within an appropriate time period.

$\mathcal{P}$:
Facing vector: $v_y^f > 0.97$.
Velocity: $|\dot{\mathbf{c}}| < 0.005$ m/sec.
Contact: hip on ground.

$\mathcal{E}$:
First phase:
  Facing vector: $v_y^f > -0.2$.
  Up vector: $v_y^u < 0.99$.
Second phase (kneel):
  Up vector: $v_y^u > 0.7$.
  Hip not on ground.
  Posture: kneeling: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.8$ rad,
  where $i = (\text{thigh}, \text{knee})$, $\mathbf{q}_0 = [-1.43 \;\; 2.92\}$,
  and n is a normalization parameter.

$\mathcal{O}$:
Up vector: $v_y^u > 0.7$.
Contact: hip not on ground.
Posture: kneeling: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.8$ rad,
where $i = (\text{thigh}, \text{knee})$, $\mathbf{q}_0 = [-1.43 \;\; 2.92\}$,
and n is a normalization parameter.
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.

The resulting motion is depicted in Figure 1.5 (b). The controller works for grounds with coefficient of friction equal or greater than 0.55.

5.8.5 Rolling over

When lying on their back, some people may choose to roll-over to a prone position before attempting to stand. We have implemented a *roll-over* controller that can emulate this action. The fourth frame in Figure 1.3 demonstrates the action of the roll-over controller. The pre-conditions of the roll-over controller require that the character be in a supine position and that the center of mass not have noticeable motion. The post-conditions of the roll controller are fairly simple; they include any prone position for which the character is extended and fairly straight; i.e., no crossing of legs or arms, etc. The expected performance is simple and makes sure that the character is facing down during the second part of the motion where it is expected to do so. The controller has a limited time period to complete its task.

$\mathcal{P}$:

Facing vector: $v_y^f > 0.5$.
Up vector: $v_y^u < 0.3$.
Velocity: $|\dot{\mathbf{c}}| < 0.005$ m/sec.
Contact: hip on ground.

$\mathcal{E}$:

Up vector: $v_y^u < 0.5$.
Velocity: $|\dot{\mathbf{c}}| < 5.0$ m/sec.
First phase:
Facing vector: $v_y^f > 0.0$.
Second phase:
Facing vector: $v_y^f < 0.0$.

$\mathcal{O}$:

Facing vector: $v_y^f < -0.5$.
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.

The controller has been tested successfully for a few different starting states. One of its limitations is the lack of synergy between different body parts. For example, when the right arm tries to pass underneath the ribs the rest of the body does not move in a fashion that reduces the weight on the arm. Therefore the arm actuators have to use more energy to compensate for the increased friction force that acts on the arm.

5.8.6 Rising from a prone position

Frames 5–9 in Figure 1.3 demonstrate the action of a controller that enables our three dimensional model to rise from the prone position. The pre-conditions require that the character be lying face down but not on his arms. The post-conditions require that the character end up in a crouching position.

$\mathcal{P}$:

Facing vector: $v_y^f < -0.3$
Up vector: $v_y^u < 0.5$.
Velocity: $|\dot{\mathbf{c}}| < 0.005$ m/sec.

$\mathcal{E}$:

Facing vector: $v_y^f < 0.5$.
Up vector: $v_y^u < 0.99$.
Velocity: $|\dot{\mathbf{c}}| < 5.0$ m/sec.
Time of completion less than 15 secs.

$\mathcal{O}$:

Balance: projection($\mathbf{c}$) $\in \mathcal{S}$.
Up vector: $v_y^u > 0.7$.

The pre-conditions are similar for the two dimensional character. However, the expected performance and the post-conditions are different, since the goal in this case is for the character to reach a kneeling state.

$\mathcal{P}$:

Facing vector: $v_y^f < 0$.
Up vector: $v_y^u < 0.5$.
Velocity: $|\dot{\mathbf{c}}| < 0.05$ m/sec.
Contact: hip on ground.

$\mathcal{E}$:

First phase:
Facing vector: $v_y^f < 0.0$.
Contact: hands on ground or hip on ground.

Second phase:
Up vector: $v_y^u > 0.7$.
Contact: knee on ground and hip not on ground.

Third phase:
Contact: Knees on ground and hip not on ground.
Posture: kneeling: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.8$ rad,
where $i = (\text{thigh}, \text{knee})$, $\mathbf{q}_0 = [-1.43 \;\; 2.92\}$,
and n is a normalization parameter.
Facing vector: $v_y^f > 0.0$.
Up vector: $v_y^u > 0.0$

$\mathcal{O}$:

Facing vector: $v_y^f > 0.0$.
Up vector: $v_y^u > 0.0$
Contact: Knees on ground and hip not on ground.
Posture: kneeling: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.8$ rad,
where $i = (\text{thigh}, \text{knee})$, $\mathbf{q}_0 = [-1.43 \;\; 2.92\}$,
and n is a normalization parameter.
Velocity: $|\dot{\mathbf{c}}| < 0.05$ m/sec.

The motion produced by this 2D controller is shown in Figure 1.6 (c). The controller is not very sensitive to friction and has been tested with a friction coefficient in the range of $[0.4, 0.6]$. Both controllers must reach their post-conditions within a limited time interval, otherwise they fail.

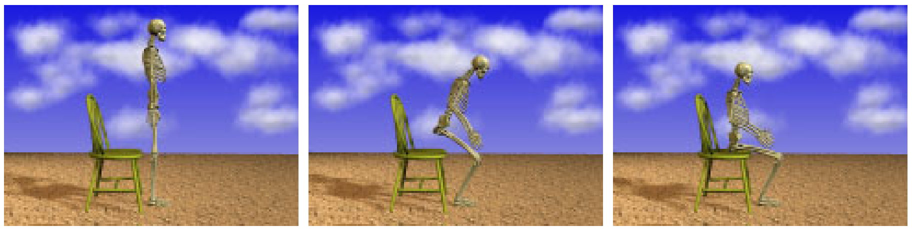
5.8.7 Kneel-to-crouch

Currently, this controller has been implemented only for the 2D model. It is a pose controller that takes the character from a kneeling position, such as the one produced by the previous controller, to a crouching position. The pre-conditions, post-conditions and expected performance are as follows:

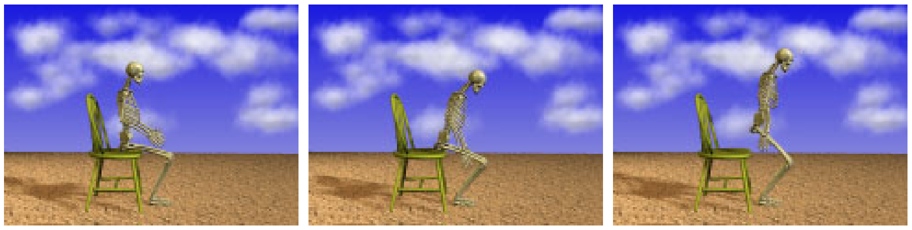
$\mathcal{P}$:

Facing vector: $v_y^f > -0.5$
Velocity: $|\dot{\mathbf{c}}| < 0.5$ m/sec.
Contact: feet and knees on ground.
Posture: kneeling: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.5$ rad,
where $i = (\text{thigh}, \text{knee})$, $\mathbf{q}_0 = [-1.43 \;\; 2.92]$,
and n is a normalization parameter.

$\mathcal{E}$:



(a) Stand to sit controller.



(b) Sit to crouch controller.

Figure 5.8: Sitting and getting up from a chair.

Velocity: $|\dot{\mathbf{c}}| < 5.0$ m/sec.
First phase:

Facing vector:	$v_y^f > -0.5$.
Contact:	foot on ground, hip not on ground.

Second phase:

Up vector:	$v_y^u > 0.5$.
Contact:	feet on ground and hip not on ground.

$\mathcal{O}$:
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.
Balance: projection($\mathbf{c}$) $\in \mathcal{S}$.

A limitation of this controller is that any rising motion before the center of mass is placed above the front foot, relies too much on the help of the back leg. A more natural approach would be to first shift the center of mass above the front foot and then have the character rise. This limitation makes the control very sensitive to the ground model.

5.8.8 Step

This is a simple controller designed for the three dimensional model that makes it perform a single step. The final stage of the controller brings both feet together to achieve a standing position. The motion is depicted in Figure 5.10. The pre-conditions require the character to be in an upright stance, while the expected performance makes sure that the character remains upright during the operation of the controller. The post-conditions require that the character is upright as the controller completes its operation.

$\mathcal{P}$:
Up vector: $v_y^u > 0.9$.

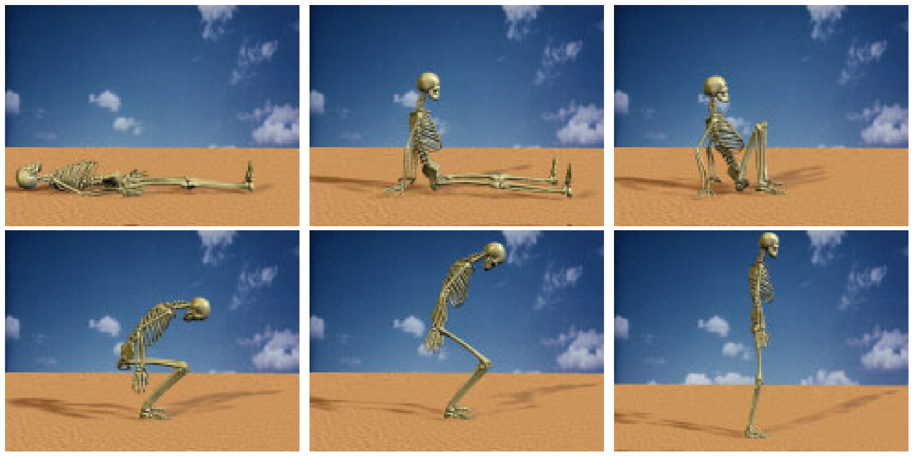


Figure 5.9: Rising from a supine position on the ground and balancing erect in gravity.

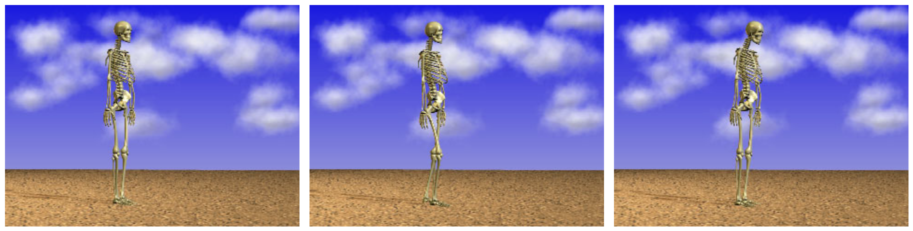


Figure 5.10: Taking a step.

Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.005$ m/sec.
Contact:	hip not on ground.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.
$\mathcal{E}$:	
Up vector:	$v_y^u > 0.9$.
Velocity:	$\lvert\dot{\mathbf{c}}_{lat}\rvert < 1.0$ m/sec,
	$\lvert\dot{\mathbf{c}}_{sag}\rvert < 0.08$ m/sec.
$\mathcal{O}$:	
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.005$ m/sec.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.

Here, $\dot{\mathbf{c}}_{lat}$ and $\dot{\mathbf{c}}_{sag}$ are the lateral and sagittal velocity of the center of mass, respectively. The controller is very sensitive to the ground model and the character model and has been tested only for a very narrow region of initial states.

5.8.9 Protective step

Human subjects whose balance is disturbed during quiet stance exhibit a variety of behaviors in their attempt to maintain balance. The sort of behavior they exhibit depends on their physical conditioning, personal preferences and the magnitude and duration of the disturbance. Pai and Patton [81] have studied under what circumstances a subject can maintain balance without stepping. Do et al. [24] have studied the biomechanical responses of human subjects to induced forward falls. They concluded that an induced forward fall starts with an invariable preparation process which is followed by an adaptable recovery one.

Our controller for the three dimensional case is currently designed to produce the visual effect. It is not sophisticated enough to actually maintain balance. However, it is designed to take step in the proper direction. Its pre-conditions are complimentary to the pre-conditions of the balance controller. At the same time they ensure the controller does not attempt to handle situations that are not appropriate. The expected performance ensures that the controller realizes its failure early so that a fall controller can take over. The post-conditions require that the character reaches a balanced upright stance.

$\mathcal{P}$:

Facing vector:	$v_y^f > -0.6$.
Up vector:	$v_y^u > 0.7$.
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 1.0$ m/sec.
Contact:	hip and hands not on ground.

$\mathcal{E}$:

Contact:	hip and hands not on ground.
Up vector:	$v_y^u > 0.7$.
Facing vector:	$v_y^f > -0.6$.

$\mathcal{O}$:

Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.05$ m/sec.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.
Contact:	hip and hands not on ground.

The three dimensional version of this controller is fairly robust in the sense that despite its failure to regain the balance of the character, it produces a satisfactory visual effect for a variety of situations, ground parameters, and human models. The motion produced can be seen in Figure 1.6 (g).

The two dimensional version of the controller is similar. However, because the character cannot fall to the side, the controller works successfully for a wide range of disturbances. The response of the controller, i.e. the length of the step that the character takes, is parameterized with respect to the velocity of the center of mass at the time that the controller takes over. When the character is pushed backwards, the final phase of the controller makes the character's torso lean forward to facilitate a potential transition to another controller. The composable interface of the two dimensional controller is as follows:

$\mathcal{P}$:

Up vector:	$v_y^u > 0.7$.
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 1.0$ m/sec.
Contact:	hip and hands not on ground.
Posture:	(standing up) $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 1.0$ rad, where $i = (\text{thigh}, \text{waist})$, $\mathbf{q}_0 = \mathbf{0}$, and n is a normalization parameter.

$\mathcal{E}$:

Facing vector:	if falling forward $v_y^f < 0.1$, if falling backward $v_y^f > -0.3$.
Contact:	hip and hands not on ground.
Up vector:	$v_y^u > 0.7$.

$\mathcal{O}$:

Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.05$ m/sec.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.

Contact:	hip and hands not on ground,
	feet on ground.
Posture:	(trunk leaning forward) $v_y^u v_y^f < 0$.

The motion produced by this controller can be seen in the last part of Figure 7.1. The protective-step controller has a higher priority than the fall controller, which ensures that, when appropriate, the character will first attempt to maintain balance by stepping and then resort to a fall behavior.

5.8.10 Crouch-to-stand

The crouch-to-stand controller achieves an upright stance starting at a variety of crouching states for both the three dimensional and the two dimensional case. The two dimensional case is more robust than the three dimensional one. However, both can fail even for states that can be considered as being "between" states that the controllers can handle successfully. The pre-conditions of both controllers ensure that the character is not already straight and that there is little movement. The composable interface is as follows:

$\mathcal{P}$:

Facing vector:	$v_y^u > -0.6$.
Up vector:	$v_y^u > 0.7$.
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 1.0$ m/sec.
Contact:	hip and hands not on ground,
	feet on ground.
Posture:	(not too straightened) $(1/n)\ \sum_i \sqrt{(q_i - q_{0,i})^2} > 0.3$ rad,
	where $i = (\text{waist}, \text{thigh}, \text{knee})$,
	$\mathbf{q}_0 = \mathbf{0}$.
	and n is a normalization parameter.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.

$\mathcal{E}$:

Contact:	hip and hands not on ground,
	feet on ground.
Up vector:	$v_y^u > 0.7$.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.
Posture:	(not too straightened) $(1/n)\ \sum_i \sqrt{(q_i - q_{0,i})^2} > 0.03$ rad,
	where $i = (\text{waist}, \text{thigh}, \text{knee})$,
	$\mathbf{q}_0 = \mathbf{0}$.
	and n is a normalization parameter.

$\mathcal{O}$:

Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.1$ m/sec.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.
Contact:	hip and hands not on ground,
	feet on ground.
Posture:	(straightened) $(1/n)\ \sum_i \sqrt{(q_i - q_{0,i})^2} \leq 0.3$ rad,
	where $i = (\text{waist}, \text{thigh}, \text{knee})$,
	$\mathbf{q}_0 = \mathbf{0}$.
	and n is a normalization parameter.

The resulting motions can be seen in Figures 1.6 (d) and 7.1.

5.8.11 Double-stance-to-crouch

This controller has been implemented for the two dimensional robot model only. It takes the character from a double stance to a symmetric one (crouch). The resulting motion can be seen in Figure 1.5 (f) and Figure 7.1. The main function of this controller is to place the center of mass above the front foot and then bring the back leg to a symmetric position. The controller is fairly robust and it has performed successfully for a variety of starting states. The composable interface is as follows:

$\mathcal{P}$:

Facing vector:	$v_y^f > -0.7$.
Up vector:	$v_y^u > 0.7$.
Velocity:	$\|\dot{\mathbf{c}}\| < 1.0$ m/sec.
Contact:	hip and hands not on ground, feet on ground.
Posture:	leg asymmetry: $\|q_i - q_j\| > 0.2$ rad, where i = left thigh and j = right thigh.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.

$\mathcal{E}$:

Contact:	hip and hands not on ground, one foot at least on ground.
Up vector:	$v_y^u > 0.7$.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.
Posture:	leg asymmetry: $\|q_i - q_j\| > 0.1$ rad, where i = left thigh and j = right thigh.

$\mathcal{O}$:

Velocity:	$\|\dot{\mathbf{c}}\| < 0.1$ m/sec.
Balance:	projection($\mathbf{c}$) $\in \mathcal{S}$.
Contact:	hip and hands not on ground, both feet on ground.
Posture:	leg symmetry: $\|q_i - q_j\| \leq 0.1$ rad, where i = left thigh and j = right thigh.
Up vector:	$v_y^u > 0.7$.

5.8.12 Walk

Walking is an essential motion for a skillful simulated agent. Unfortunately it is also a difficult motion to simulate in a robust fashion. Despite the large amount of research work on dynamic walking, there are no solutions that are sufficiently general with respect to different models and terrains.

Our controller for two dimensional walking implements a slow walking gait over flat terrain. The idea behind this controller is to use the swing leg to bring the center of mass above the pivot leg and then swing while the center of mass is statically balanced. When the swing leg is in front of the body, the controller throws the character on the swing leg (now the front leg) using the ankle of the back leg. Frames from an associated animation can be seen in Figure 7.1. The user can specify how many steps the character must take. The controller maintains a step counter and when the required number of steps is reached, it signals success, leaving the character in a double stance position (feet spread). The composable interface is as follows:

$\mathcal{P}$:	
Facing vector:	$v_y^f > -0.5$.
Up vector:	$v_y^u > 0.9$.
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.5$ m/sec.
Contact:	hip and hands not on ground, feet on ground.
Posture:	upright: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 0.1$ rad, where $i = (\text{thigh}, \text{waist}, \text{knee})$ and n is a normalization parameter.
Balance:	projection$(\mathbf{c}) \in \mathcal{S}$.
$\mathcal{E}$:	
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 1.0$ m/sec.
Contact:	hip and hands not on ground, one foot at least on ground.
Up vector:	$v_y^u > 0.7$.
Balance:	projection$(\mathbf{c}) \in \mathcal{S}$.
$\mathcal{O}$:	
Velocity:	$\lvert\dot{\mathbf{c}}\rvert < 0.1$ m/sec.
Balance:	projection$(\mathbf{c}) \in \mathcal{S}$.
Contact:	hip and hands not on ground, both feet on ground.
Up vector:	$v_y^u > 0.7$.
Required number of steps reached.	

The controller is robust with respect to a narrow region of initial upright stance configurations and it can tolerate small changes in the ground friction coefficient.

5.9 Stunts

In addition to everyday actions, our dynamic character should be able to perform a variety of interesting voluntary actions dictated by the animator. Such actions can potentially include physically dangerous stunts.

5.9.1 The *kip* move

The *kip* stunt, shown in Figure 5.11, is an athletic motion often seen in martial arts films. It provides a very quick way to get up from a supine position. Variations of the kip are used extensively in gymnastics, Bergemann [11]. The basic mechanics of the motion are not very complex but the timing is crucial. The main idea behind the kip is to get the body airborne and at the same time provide enough rotational momentum to allow the feet to be placed under the center of the mass. We determined the mechanics of the kip by observing human subjects, as shown in Figure 5.11. The maximum height of the center of mass during a kip varies between subjects. Subjects with a gymnastics background tend to have a smoother and more gracious motion such as the one performed by the human subject in Figure 5.11. Subjects with a martial

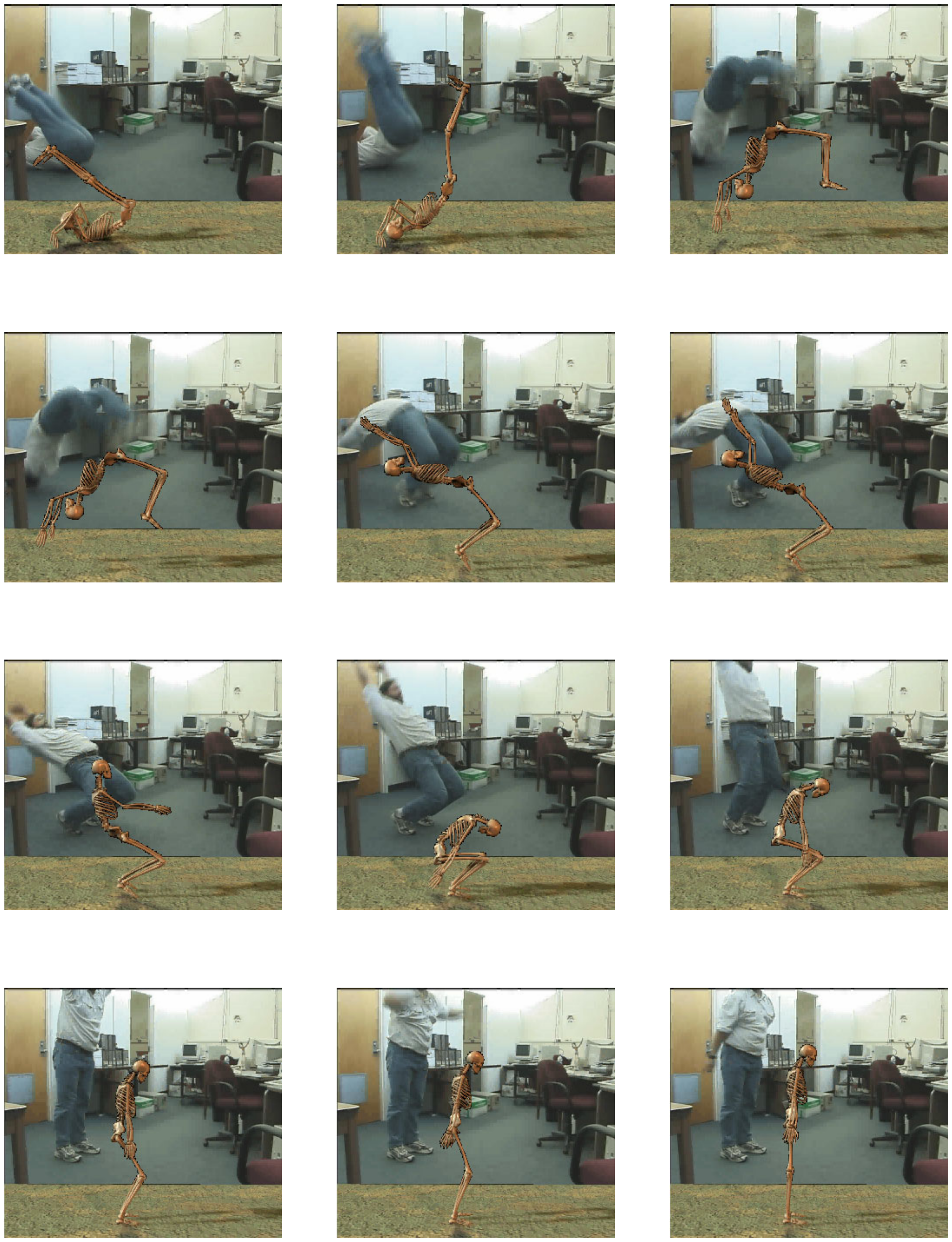


Figure 5.11: The *kip* move performed by both a real and a virtual human.

arts training tend to perform a sharper motion with less height and a landing posture closer to the crouch posture of the simulated character.

The controller is based on a pose controller whose pre-conditions include a variation of supine positions. The first pose of the controller makes sure that the character assumes a position suitable for performing the kip. This include straightening the legs and arms. The larger part of the motion is ballistic, which focuses the control mainly at the kick off and landing phases. The last part of the controller applies continuous control to bring the stuntman to an erect position from which the balance controller can take over. Alternatively, the kip controller could finish with the character in a crouched position and let another controller perform the crouch-to-stand action, such as the *crouch-to-stand* controller presented in Section 5.8.10. The specifications below correspond to a kip controller that leads to a crouching position. The expected performance of this controller is relatively simple because the motion is fast and largely ballistic.

$\mathcal{P}$:

Velocity: $|\dot{\mathbf{c}}| < 0.005$ m/sec.
Face up: $v^f_y > 0.97$.
Posture: lying on back: $(1/n)\sum_i \sqrt{(q_i - q_{0,i})^2} < 1.5$ rad,
where $i = (\text{thigh}, \text{knee}, \text{waist})$, $\mathbf{q}_0 = \mathbf{0}$
and n is a normalization parameter.
Contact: hip on the ground.

$\mathcal{E}$:

Horizontal Velocity: $|\dot{\mathbf{c}}_h| < 1.0$ m/sec.
Requires that the character doesn't lean sideways, $|v^u_z| < 0.05$.

$\mathcal{O}$:

Up vector: $v^u_y > 0.7$.
Velocity: $|\dot{\mathbf{c}}| < 0.1$ m/sec.
Balance: projection$(\mathbf{c}) \in \mathcal{S}$.
Contact: feet on the ground, hip not on the ground.

Our kip controller is very sensitive to the ground model. Implementing a more robust version of the controller would be an interesting short project. The rigid back of our articulated model, was a limitation during the development of the kip controller. The kip requires the character to roll on its back, and a flat back makes this difficult. Our controller spends more energy than one would expect while performing the initial rolling back motion.

5.9.2 Plunging and rolling

Plunging down stairs or slopes is not a task that any human subject would like to perform for the purposes of motion capture. Such dangerous stunts are better left to simulated characters. Figure 5.12 shows the stuntman performing a suicidal dive down stairs. The character can be instructed to lunge forward and upward at an angle specified by the user. When the hands contact the ground or 2 seconds after the last pose becomes active, the controller assumes success. This allows another controller to take over, and handle the impact with the ground, for example a gymnastic controller that can absorb the shock in a specific fashion. If such a controller does not exist then the default controller takes over and the character rolls above his head. The plunge controller bids for control of the character only when dictated by the user. For this reason and because the duration of the action is short, the pre-conditions, post-conditions

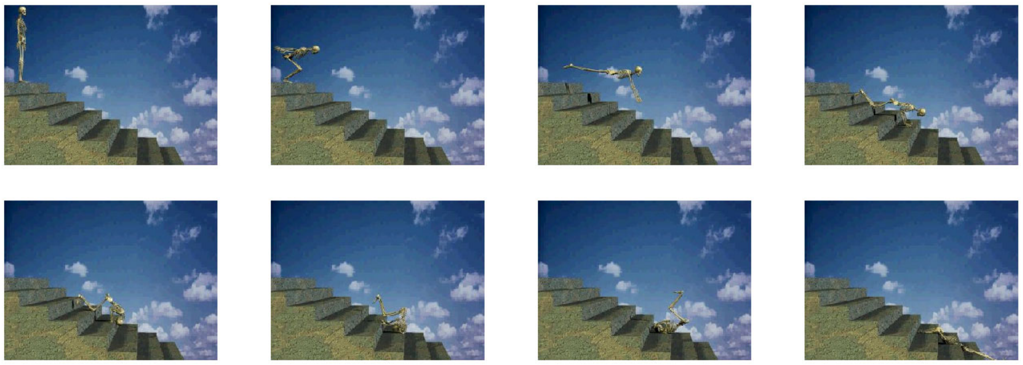


Figure 5.12: Ouch!

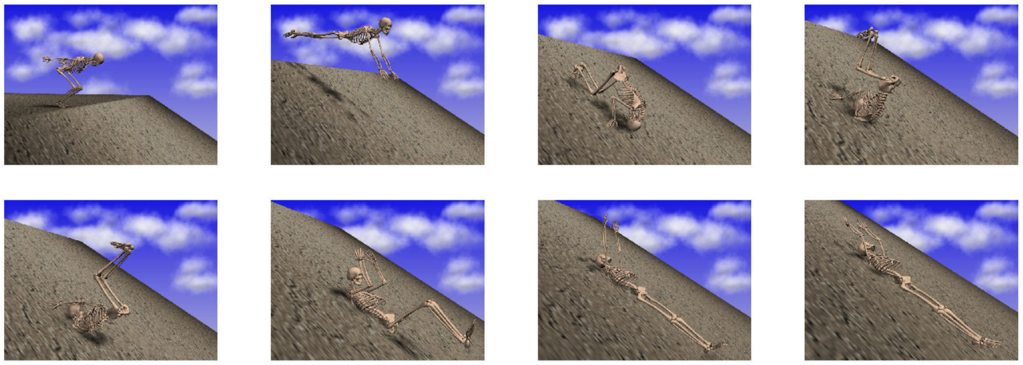


Figure 5.13: Plunge and roll on a different terrain.

and expected performance of this controller are relatively simple. The expected performance ensures that the character falls with gravitational acceleration until its hands touch the ground.

$\mathcal{P}$:
Velocity: $|\dot{\mathbf{c}}| < 0.01$ m/sec.
Face up: $-0.5 < v_y^f < 0.1$.
Contact: feet on ground, hip not on ground, hands not on ground.
Balance: projection($\mathbf{c}$) $\in \mathcal{S}$.
$\mathcal{E}$:
Before take off:
Leaning forward less than the user specified angle.
After take off:
Acceleration: $\ddot{c}_v < -9.8$ m/sec^2.
Contact with the ground in 2 seconds.
$\mathcal{O}$:
Contact: hands on ground.

Actions such as plunge and roll are well suited for a physics-based solution. The ballistic part of the motion and the collisions with the ground are solved automatically by simulating the effects of gravity and the collision forces. Although skilled animators are able to design accurate kinematic solutions for such motions, it requires considerable effort and skill to produce kinematically the illusion of mass and the transfer of momentum. In addition, kinematic solutions must be manually adjusted to novel terrains. In contrast, physics-based controllers can likely be re-used for similar terrains. Figures 5.12 and 5.13 show the same plunge and roll controller operating on two different terrains.

5.10 Discussion

The composable interfaces of the controllers presented in the previous section have been developed based in part on information from biomechanics, on our intuition, and on experimentation. They have been applied successfully in our experiments as described in Chapter 7. The computation of particular virtual sensors such as posture, kinematic characteristics of the center of mass etc made the design of the above controllers easier. Often, there are redundant ways to represent the same sensor information. For example, the length of the support polygon and the angles at the leg joints, can both show whether the feet are in a symmetric position or not. Choosing between redundant representations is up to the designers of the controllers.

It is worth noting that while the character is described by a well defined state vector which is part of the feedback information that controllers use, the environment does not have an equivalent state description. We do not currently have a definition of the *state* of the environment that can be combined with the state of the character and form an augmented state space which can then serve as the domain of the pre-conditions. Determining the parameters that completely define the state of the environment is a difficult task and is part of our future work. The development of physics-based controllers has given us an opportunity to see how interesting and complex the human body is. It is surprising how difficult it is to describe in a robust and algorithmic fashion the muscle activations that realize these motions. It is interesting to note that the difficulty of actually performing a motion and simulating it are unrelated. Consider the following question: Which one of the following two controllers is it more difficult to design by hand, the *kip* controller or the *crouch to stand*? Surprisingly the *kip* controller took four

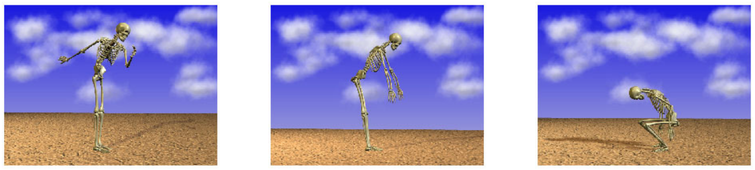


Figure 5.14: Different crouching configurations.

hours to implement, while the *crouch to stand* took two days. We attribute this difference to the quick and ballistic nature of the kip in addition to its well-defined starting state. Most of the effort went into designing the kick off stage. On the other hand, despite the fact that the *crouch to stand* controller implements a common everyday motion that is performed in place, the character must be balanced for the duration of the motion. At the same time the controller has to handle drastically different initial configurations as shown in Figure 5.14.

The controllers that have been presented in this chapter are mostly designed to serve as examples of the composition scheme. They can be improved in a variety of ways to become more robust in terms of the initial configurations they can handle. We believe that our composition framework along with the base system presented in Chapter 6, can form the basis of a common animation system through which researchers and students can build upon and improve existing results. Chapter 7 discusses the results that we were able to achieve using the above controllers and the composition framework described in Chapter 3.

Chapter 6

Dynamic Animation and Control Environment

This chapter describes the animation system that we have developed jointly with Victor Ng-Thow-Hing. The *Dynamic Animation and Control Environment* (DANCE) is a flexible animation software package that we believe can become the common tool for animation research.

6.1 Motivation

Physics-based animation (PBA) has often been touted as a solution for automatically creating realistic motion. Yet this technique remains underutilized in many commercial animation systems. Indeed, software packages[1] that feature 3-D dynamics often restrict the methods of active force generation on the objects, allowing only passive simulation. Motion synthesis with PBA often requires the animator to set up several physical parameters and passively watch the resulting simulation. Usually, multiple iterations are required to achieve a satisfactory animation. This motivates the need to allow the animator to interactively control the simulation.

An interesting area of research in PBA is the design and construction of *controllers* that can compute the active forces and torques required to control a character, either as an articulated figure or a deformable object. Controllers have been created to produce a range of locomotive tasks, such as swimming[99], running[47] and walking[62]. Although controllers are inherently reusable, their actual implementations were often restricted and embedded in custom systems built by various research groups for their own specific purposes. Other experimental systems only allowed a limited number of different types of controllers to coexist, Tu [99]. The incorporation of new controllers into such systems often implied a large undertaking in code redesign and development. The difficulty of sharing simulators and controllers is an impeding factor in the advancement of physics-based animation techniques, especially considering the difficulty of designing robust controllers for complex characters.

DANCE allows controllers to be built as separate programs that adhere to a specific interface using standard object-oriented design. The end result is that an animation can be constructed consisting of several controllers cooperating or competing with each other.

6.2 Features

The main feature's of DANCE are:

- *Modularity.* DANCE's object-oriented and modular architecture allows the re-use of a variety of components.
- *Portability.* The use of widely available libraries such as Tcl/Tk, OpenGL/GLUT allows DANCE to run on Win95,Win98,WinNT, Win2000, IRIX, and LINUX systems. An earlier version also ran on MacOS 8.
- *Flexibility.* DANCE's plug-in architecture allows the user to load dynamically linked objects at run time.
- *Script-based interface.* The full command set of the Tcl script language is available in DANCE.
- *Customizable user interface.* The graphical user interface of DANCE is based on Tk and it can be fully customized with Tk scripts.
- *Multiple views.*. The user can create any number of customized views of the scene. During simulation the contents of the views can be saved as sequences of images to create movie clips.

DANCE employs the following widely available libraries which are included with the DANCE distribution[1] for convenience.

Arcball Ken Shoemake[43] introduced the intuitive Arcball technique for rotating camera views. The original sample code provided uses the venerable IrisGL graphics API. This code has been modified to work with the standardized OpenGL. In addition, new functions were added that allow the camera matrix to be initialized to an arbitrary rotation matrix.

f2c Several numerical routines used by various plug-ins in DANCE were converted from FORTRAN to C *f2c* which was created by AT&T, Lucent Technologies and Bellcore.

OpenGL 3-D graphics are handled by the OpenGL API, including selected functions from the GLU library. OpenGL was chosen over other APIs because of the cross-platform requirements of our design.

GLUT Mark Kilgard's GLUT library provides the windowing system and the main loop for DANCE. Mouse and keyboard events are also handled by the GLUT callback mechanism. The version of the GLUT library used in DANCE has been modified slightly to handle shared OpenGL display lists amongst several windows.

Minpack The Minpack library provides useful routines for unconstrained optimization. Several plug-ins use this library created by Jorge Moré, Burt Garbow, and Ken Hillstrom at the Argonne National Laboratory.

RAPID RAPID provides the base for efficient collision detection and proximity tests. RAPID's authors can be contacted through S. Gottschalk from the Department of Computer Science at Chapel Hill.

[1]DANCE is available at http://wwww.dgp.toronto.edu/DGP/software/dance/dance.html.

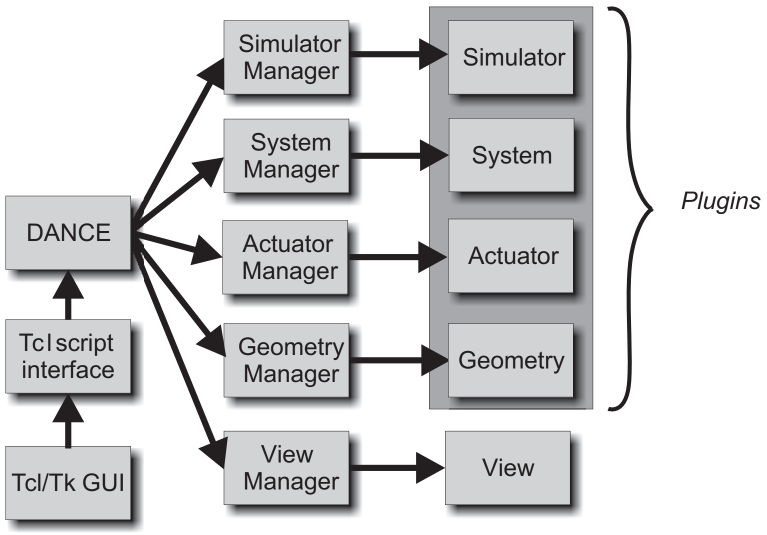


Figure 6.1: The architecture of DANCE.

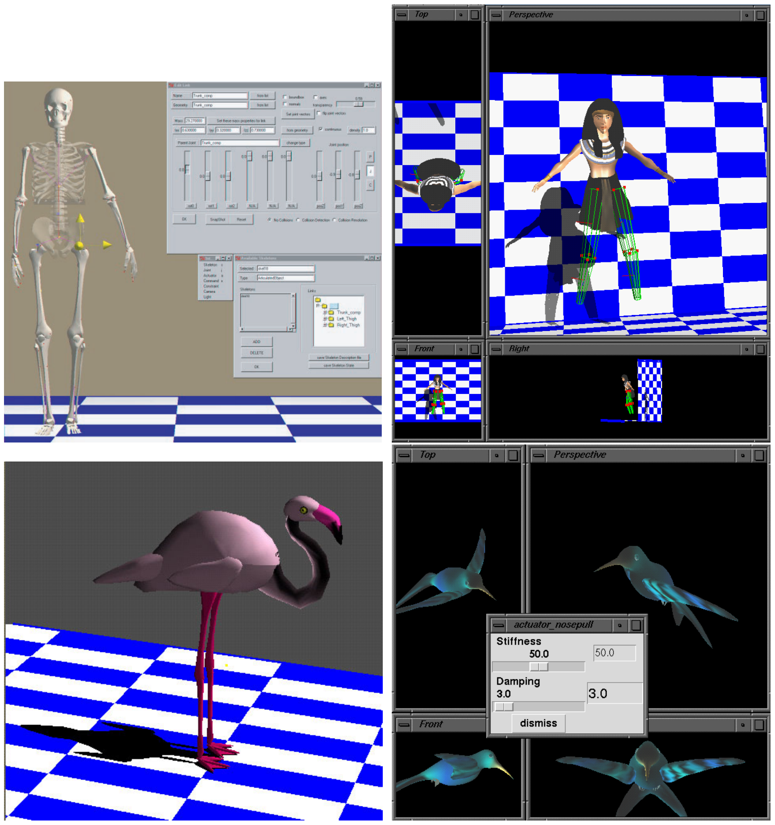
Tcl/Tk	The scripting engine provided by the DANCE core is based on the Tcl language provided by Scriptics Corporation. In addition, the choice of Tcl enables graphical user interfaces to be created with Tk, which is an extension of Tcl.
TkCon	The enhanced Tk Console written by Jeffrey Hobbs and with contributions from numerous others serves as a great UNIX-like console window.
VCOLLIDE	To track multiple body collisions, the team from the Department of Computer Science at Chapel Hill created VCOLLIDE which works with RAPID. The ArticulatedObject plug-in uses VCOLLIDE to keep track of all the geometry assigned to the links of every articulated object in DANCE.
VRMLView	We use VRMLview, created by Systems in Motion, for previewing VRML models for one of our geometry plug-ins.

6.3 Component abstraction

We aim to make DANCE the unifying platform in animation research where researchers can build on top of the work of others or simply use it for whatever purpose serves them. Thus, one of DANCE's design goals is to provide *application programming interfaces* (APIs) that can capture almost any virtual environment. During the design phase, we spent a significant amount of time trying to identify and refine those components that provide the required flexibility and generalization to achieve this goal. The resulting components are *systems, simulators, actuators, geometries and views.* As shown in Figure 6.1 only the first four are plug-ins.

6.3.1 Systems

A *system* captures passive or active entities that can move or simply exist in a virtual environment. Such entities may represent characters, inanimate objects, particle systems, and generally any kind of physical system. Systems typically have degrees of freedom, $\mathbf{q}$, that coupled with



Stiffness 50.0
Damping 3.0

Figure 6.2: Articulated figures in DANCE.

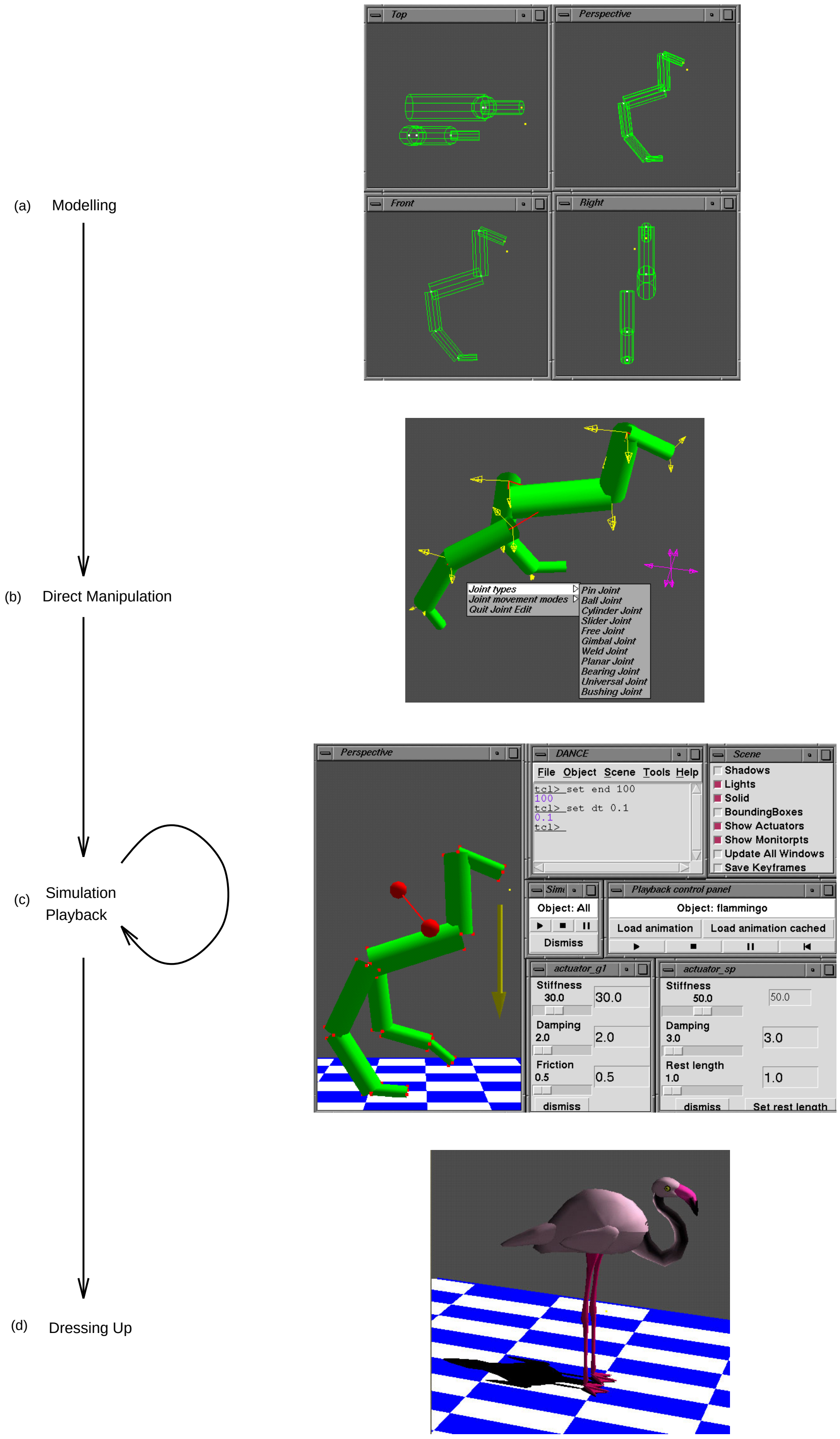
their velocities represent the system's kinematic state, $[\mathbf{q}\ \ \dot{\mathbf{q}}]^T$. The `System` base class provides a general API that allows researchers to implement a wide variety of systems. To date, the following systems have been implemented as DANCE plug-ins.

Articulated figures

The `ArticulatedObject` plug-in provides a reduced coordinate, body-joint representation for articulated systems without loops. It supports a wide range of joint types such as free, ball, pin, gimbal and universal. Joints parameters and the associated degrees of freedom can be directly manipulated by the user in a forward kinematics fashion. The user can interactively assign geometry to the links for visual effect. During forward kinematics manipulation, limited collision detection and resolution is provided for links with geometry. Figure 6.2 shows a variety of systems that have been modeled using the *ArticulatedObject* plug-in. The user can interactively design an articulated figure as shown in the work flow of Figure 6.3, dress it up with geometry and animate it.

Flexible characters

Deformations play an important role in computer graphics, allowing mundane objects to assume interesting new shapes. The free-form deformation (FFD) [95], essentially a geometric spline deformation that is manipulable through a lattice of control points, is a popular tool for modeling and animating nonrigid objects. Our formulation [27] generalizes conventional FFDs



```
tcl> set end 100
100
tcl> set dt 0.1
0.1
tcl>
```

[ ] Shadows
[✓] Lights
[✓] Solid
[ ] BoundingBoxes
[✓] Show Actuators
[✓] Show Monitorpts
[ ] Update All Windows
[ ] Save Keyframes

Object: All

Object: flammingo

actuator_g1: Stiffness 30.0 — 30.0; Damping 2.0 — 2.0; Friction 0.5 — 0.5

actuator_sp: Stiffness 50.0 — 50.0; Damping 3.0 — 3.0; Rest length 1.0 — 1.0

Figure 6.3: Working with articulated figures in DANCE.

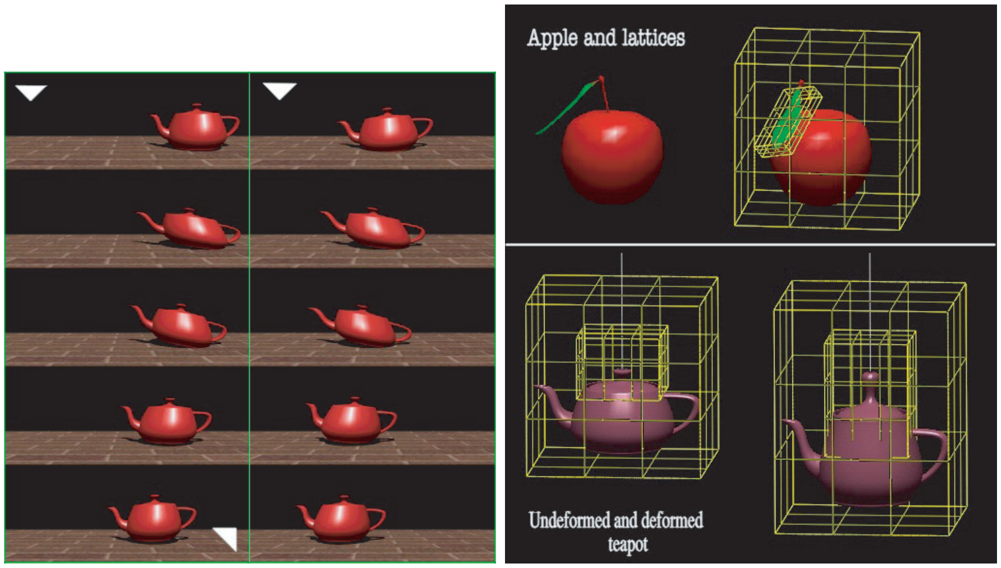


Figure 6.4: Dynamic free-form deformations in DANCE.

within a dynamic setting. Dynamic FFDs offer animators the opportunity to apply physical dynamics to the creation of expressive, cartoon-like animators. Our dynamics FFD model conforms to DANCE's *System* API and can share a virtual environment with other systems such as articulated figures. Figure 6.4 shows examples of flexible characters within the associated deformation lattice and frames from an animation created with DANCE.

Particle systems

Particle systems were first introduced in graphics by Reeves [89]. Since then, they have been used in variety of applications ranging from modeling fire, explosions, rain, clouds and other natural phenomena to modeling hair, fur, melting solids, deformable objects, landscapes and textures.

The particle system implemented in DANCE models a relatively simple case. Particles are generated from a flat source whose position can be animated. The particle direction and initial velocity can be stochastically distributed around user specified values. Their birth rate and life time are also specified by the user. Particles can have a variety of parameterized shapes and mass properties. Using this particle system we have simulated rain, snow and a variety of simple phenomena. Figure 6.5 shows a snapshot from a simulation involving two systems, a two-link pendulum modeled using the `ArticulatedObject` plug-in and a particle system whose source is attached to the lower link of the pendulum.

6.3.2 Simulators

The state of systems can change directly or under the influence of forces. The former method is typical of kinematic techniques that compute the position and velocity parameters of a system either procedurally or by manipulating parameterized motions curves [111, 13]. Physics-based techniques simulate the physical interaction between a system and its environment. For such techniques, motion is a result of applied forces. For these cases, the module that defines the evolution of a system's state through time is called a *simulator*. DANCE provides a simulator

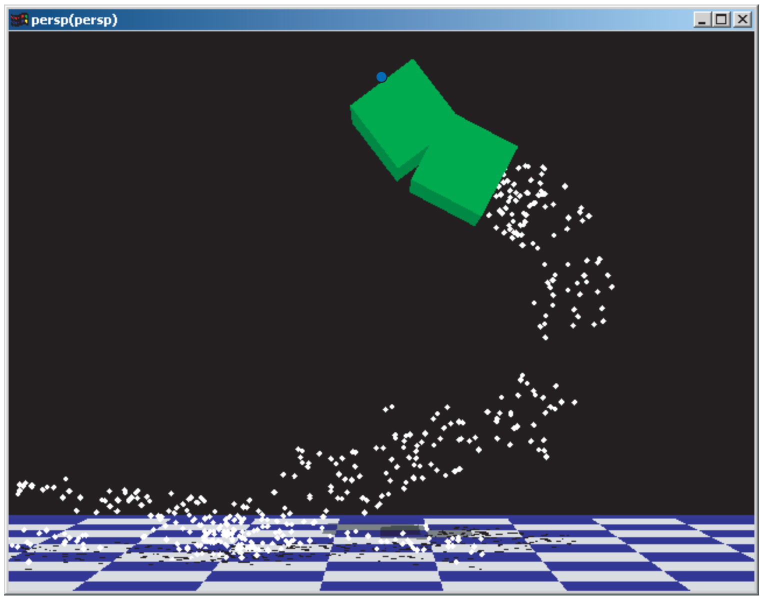


Figure 6.5: A two-link ...saltshaker.

API through the base class `Simulator` which can be used to model both kinematic and physics-based simulators.

Conceptually each system can have its own simulator which, in the physics-based case, describes how the kinematic state of that system changes under the influence of forces. In the case of interacting dynamical systems this creates consistency and synchronization problems since action-reaction forces and contact information may be inconsistent between simulators [8]. Systems of the same type can be handled by a common simulator simultaneously which eliminates the consistency problems. Our current implementation of multiple simulators uses a layered approach which in some cases can have consistency problems. In the future we would like to implement a method that handles multiple simulators properly and allows parallel simulation on multiprocessor machines.

Articulated figure simulator

There is a vast amount of work within the robotics literature, dealing with the dynamics of articulated systems. Featherstone [29] is a classic reference for articulated figure algorithms while Goldstein [37] presents a detailed description of classical mechanics. Cerezo et al. [17] present a comprehensive survey of animation and simulation techniques, new trends, groups that are active in the area and extensive bibliographical data.

There is a variety of simulation software that can be used to simulate articulated figures. Mirtich [74] developed an impulse-based simulator. Faure [28] implements an interesting iterative method that allows the user to balance accuracy with efficiency. Baraff [7] presents a linear time version of Lagrange dynamics. Mathengine is a commercial simulator with a focus on gaming applications. SD/Fast is a commercial simulator generator that uses a reduced coordinate formulation to produce very efficient simulation code.

Within DANCE we have implemented an articulated figure simulator based on SD/Fast. SD/Fast takes the description of an articulated figure and produces an optimized version of Equation 5.1 in C. Our simulator discretizes the time in small steps and for each step it uses code generated by SD/Fast to compute Equation 5.1. The equations of motion are solved using Kane's $O(n)$ method to yield the acceleration of the character's degrees of freedom $\ddot{\mathbf{q}}$. At the end of each time step, the latter are integrated twice, using a fourth order Range-Kutta method, to produce the new positions, $\mathbf{q}$ and velocities, $\dot{\mathbf{q}}$. The version of DANCE running under the Linux operating system automates the production of the equation of motions. As the simulation starts our plug-in simulator can automatically call SD/Fast[2] to produce the equations of motion, and then compile and link the produced code to the running DANCE session.

Lagrange simulator for DFFD system

The dynamic flexible system presented in Section 6.3.1 is associated with its own simulator that describes how external and actuator forces affect its kinematic state. To derive the equations of motion we employ a Lagrange formulation [69]. In contrast with the Newtonian approach which is based on the relationship between applied forces and the momentum of a system's degrees of freedom, the Lagrange formulation is based on the relationship between the energy of a system and its degrees of freedom. This simulator is described in detail in [27].

Particle system simulator

The motion of the particle system described in Section 6.3.1 is computed analytically because of its simple structure. The velocity and position of each particle, i, is given by the following simple equations:

$$\mathbf{v}_i(t) = \mathbf{v}_{o,i} + \mathbf{g}\Delta t, \tag{6.1}$$

$$\mathbf{p}_i(t) = \mathbf{p}_{o,i} + \mathbf{v}_i\Delta t + \frac{\Delta t^2}{2}\mathbf{g}, \tag{6.2}$$

where the $\mathbf{g}$ is the acceleration of gravity, typically $(0, -9.8, 0)$. In the current implementation, the collision of a particle with a horizontal ground is hardcoded in the simulator and handled as follows:

$$v_{i,y} = -v_{i,y} lossFactor, \tag{6.3}$$

$$v_{i,x} = v_{i,x} + \texttt{rand}(-0.5, 0.5), \tag{6.4}$$

$$v_{i,z} = v_{i,z} + \texttt{rand}(-0.5, 0.5), \tag{6.5}$$

$$p_{i,y} = grHeight, \tag{6.6}$$

where $lossFactor$ is associated with the amount of the kinetic energy that a particle loses at each collision with the ground, $\texttt{rand}(-0.5, 0.5)$ produces a random number in the interval $(-0.5, 0.5)$ that when added to the horizontal component of a particle's velocity simulates the effect of a non-perfectly smooth surface and $grHeight$ is the height of the ground.

We have also incorporated McAllister's particle system API [67] into DANCE. To do this we have separated McAllister's code into two wrapper plug-ins, one *system* and one *simulator*. McAllister's API is a very powerful tool for creating elaborate particle system animations.

[2]Assuming that a valid commercial license has been installed.

6.3.3 Actuators

The simulators we have described above define the motion of the associated systems as a result of applied forces. This section describes how these forces are implemented and incorporated in a DANCE simulation.

With respect to a given system, we can distinguish between *internal* and *external* forces. Internal for us are those that a system can exert using its actuators, and external forces are those that other systems can exert on it. For example, simulated characters can exert internal forces using their virtual muscles and they can be affected by external forces such as ground reaction forces and gravity. In DANCE, anything that can exert a force, either internal or external, is captured by the `Actuator` API. This API allows actuators to control other actuators and form hierarchies that can be as complex as the programmer desires.

We have developed a variety of useful actuators within DANCE which we present in the next sections.

Force Field

This is a generic force-field actuator whose direction and magnitude can be interactively manipulated by the user. To date we have used it to simulate gravity and constant force fields.

Spring Forces

The `SpringForce` actuator allows the user to interactively attach springs anywhere on an object. Once both ends of the actuator are attached a spring force comes into effect to maintain the rest length of the spring actuator. The forces developed on the object at the points $\mathbf{p}_1, \mathbf{p}_2$ where the spring is attached have the form:

$$\mathbf{F}_{1,sp} = -\mathbf{F}_{2,sp} = -K([\mathbf{p}_1 - \mathbf{p}_2] - L_{rest}\frac{\mathbf{p}_1 - \mathbf{p}_2}{||\mathbf{p}_1 - \mathbf{p}_2||}) - D(\dot{\mathbf{p}}_1 - \dot{\mathbf{p}}_2), \tag{6.7}$$

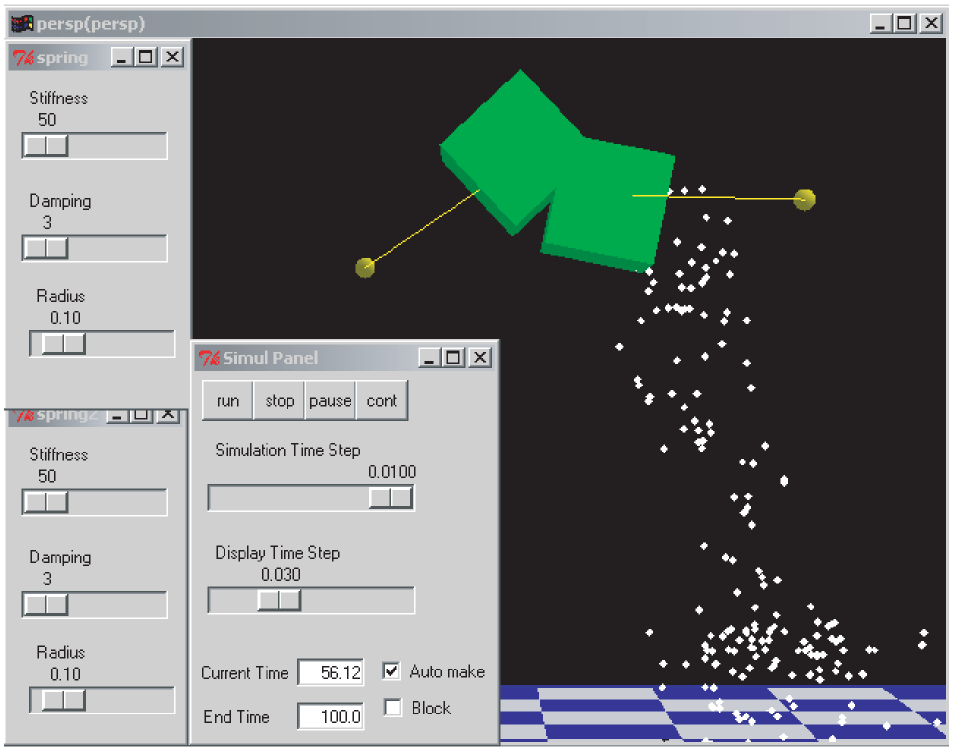
where L_{rest}, K and D are the spring's rest length, stiffness and damping constants. The user can change these constants interactively. Figure 6.6 shows two spring actuators holding the pendulum of Figure 6.5 in place during a simulation. The graphical user interfaces of the spring actuators and the simulator are also shown.

Point Force Actuator

Point forces are a convenient way to model contact between the user and objects in the environment. We have used point forces to push on simulated characters, and move objects. In the current implementation, the user can interactively apply a force actuator at any part of an object. The application point is always the center of mass of the associated part of the object, although it would be trivial to allow arbitrary application points. In addition, the user can interactively specify the magnitude and direction of the applied point force.

Collision Actuator

In the context of animation, moving objects may collide with each other. It is therefore essential for an animation system to detect collisions and resolve them in some way. Physics-based animation typically considers the motion of objects at discrete points in time separated by either a constant or variable time step. By keeping the time step small a better approximation



Stiffness
50

Damping
3

Radius
0.10

Stiffness
50

Damping
3

Radius
0.10

Simulation Time Step
0.0100

Display Time Step
0.030

Current Time 56.12

End Time 100.0

[✓ Auto make]

Block

Figure 6.6: Working with articulated figures in DANCE.

of the continuous motion of the objects is achieved. When collisions are involved this time step must be really small to avoid missing collisions between objects and having objects pass through each other between time steps. Collision detection is often the most computationally expensive part of a simulation.

Collision detection between polyhedral objects has been of great interest in a variety of disciplines including robotics, graphics and computational geometry. Most methods use bounding box techniques to quickly identify which parts of the objects involved are in collision, and for those parts they accurately compute the points, polygons or surfaces that are in contact. Our actuator uses the RAPID library which implements an efficient collision detection technique based on *oriented bounding boxes* [38]. For applications where a large number of moving objects might collide with each other it is useful to exploit the temporal coherence of the motions involved to increase the efficiency of the collision detection method. A number of collision detection packages employ this idea such as V-COLLIDE [40] and V-Clip [75].

After a collision is detected the system has to ensure that the colliding objects will not penetrate each other. The most accurate way to solve this problem is to compute the exact time of collision and apply an impulsive response that prevents the objects from interpenetrating [74, 9]. However, computing the exact time of collision and handling impulsive forces is relatively complicated and it can be a problem for interactive applications. The most popular (but not the most stable) method of resolving collisions for animation applications is based on the *penalty* method. When objects penetrate each other a spring force that depends on the amount of penetration comes into effect and pushes the objects apart. Compared to methods based on impulses, penalty methods are easier to implement and they can simulate resting contact more easily. However, they produce stiff systems of equations which reduce the simulation speed.

Our actuator works for polygonal objects represented by sets of triangles. In particular, it

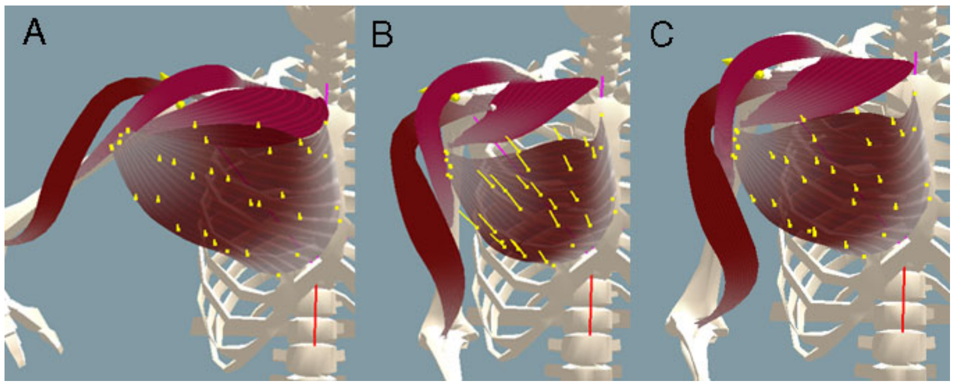


Figure 6.7: A complex muscle actuator, courtesy of Victor Ng-Thow-Hing.

can work for any system whose geometric representation is provided by the `IndexedFaceSet` plug-in described in Section 6.3.6. Before all simulators take a step, and for each pair of objects, the collision actuator registers all the vertices that are in collision and computes the associated collision forces. The vertices and the forces are stored in a hash table for fast access. When each simulator takes a step, the collision actuator retrieves the collision forces of the object under simulation and passes them to the simulator for application. Our actuator is based on a penalty method using a linear spring model. When a vertex penetrates a triangle, the actuator keeps track of the point of entry and pushes the objects apart with a force proportional to the interpenetration. The current implementation models only sliding friction. In the future we would like to implement a proper Coulomb friction model.

6.3.4 Ground actuator

Most simulated environments necessarily include a model of the ground the characters can stand on. Our `PlaneGround` actuator plug-in can model any piece-wise linear terrain. Collisions between a system and the ground are detected through monitor points that the system provides. Monitor points are defined and accessed through an API provided by DANCE to all systems. Thus, the PlaneGround actuator is fairly generic. Collision forces are based on a penalty method that implements a Coulomb friction model. For every point that penetrates the ground a reaction force takes effect at the entry point that models both sliding and static friction using friction cones. If for a given monitor point, the computed friction force falls out of the friction cone then its initial point of entry in the ground is adjusted to correct for it.

6.3.5 Musculotendon model

Victor Ng-Thow-Hing's Ph.D. thesis [77] is an example of a very complex plug-in actuator for DANCE. Ng-Thow-Hing develops a biomechanically accurate muscle model based on B-spline solids that can undergo volume preserving deformations. The muscles can be attached to a skeleton interactively and follow the skeleton's motion. A Lagrangian simulator allows the muscles to exert forces on the skeleton, accurately modeling real muscles. A customized collision detection and resolution module allows the user to design elaborate muscle groups that are in contact with each other or with the bones of the skeleton, as shown in Figure 6.7.

Inverse Kinematics Actuator

Animating complex articulated structures can be tedious because of the large number of parameters that animators have to adjust in order to produce a desired motion. *Inverse kinematics* is a technique that comes from robotics research and it can reduce the number of parameters that an animator has to control during an animation.

Considering that an articulated system is hierarchical structure of links connected with joints, *Forward kinematics* propagate the motion of parent links to the child links. *Inverse kinematics* (IK) do the opposite. A motion is applied to a child link which is called *end effector* and the IK system automatically propagates this motion up the hierarchy. Inverse kinematics methods are used extensively in the animation of articulated figures and they are a standard tool in most commercial animation tools.

The IK actuator in DANCE has been implemented by Gordon Cook as a summer project. It allows the user to set position and orientation constraints for multiple end effectors interactively. The IK engine attempts to satisfy the constraints iteratively without an explicit calculation of the Jacobian [64].

Physics-based Controllers

The controllers described in Chapter 5 are all plug-in actuators for DANCE. They implement a variety of control methods including finite state machines and continuous feedback control. In addition, the composition framework presented in Chapter 3 is a nice example of an actuator controlling other actuators in a hierarchical fashion.

6.3.6 Geometries

Most DANCE systems represent a physical entity which has a geometric representation. Actuators may also have a visual representation for purposes of direct manipulation. These representations are loaded into DANCE through the `Geometry` API. This API allows plug-ins to define a number of interesting functionality such as monitor points for collision detection, triangle structures used by RAPID, and routines to compute mass properties. In addition, the geometry API allows systems and actuators to share the same geometry plug-ins to reduce computer memory usage.

We have implemented only one geometry plug-in so far, called `IndexedFaceSet`. It can represent arbitrary polygonal objects using an indexed face set structure and it can read subsets of VRML, OBJ, and PLY file formats. When an object is loaded into an `IndexedFaceSet` plug-in, polygons are automatically transformed into triangles for use with RAPID. If specified by the user, `IndexedFaceSet` can automatically compute the mass of the object and a diagonal inertia tensor using a method proposed by Mirtich [73].

6.4 Implementation

DANCE is an object-oriented modular software package. Its class hierarchy is shown in Figure 6.8. The top class, `DanceObject`, implements a generic API that every class within DANCE needs. This API includes object identification (name, type), bounding box definition, and hooks to the main loop, the simulation loop, and the user interaction module (direct manipulation, keyboard callbacks etc). Class `PlugIn` implements the necessary API for dynamically loaded plug-ins. The rest of the classes work as follows:

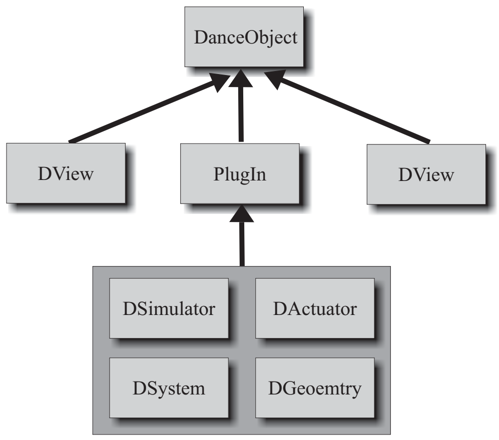


Figure 6.8: Class hierarchy in DANCE.

Software module	Without DANCE	With DANCE
Base system: console, interface, components	4 months	0
Articulated figure and simulator class	5 months	0
Actuators: Collision, ground, playback etc	2 months	0
Miscellaneous: Utilities, Optimization, debugging	1 month	0
Getting familiar with DANCE	0	3 hours - 2 days
Total	13 months	3 hours - 2 days

Table 6.1: DANCE can save time.

- `DSystem`. Implements the system API.
- `DSimulator`. Implements the simulator API.
- `DActuator`. Implements the actuator API.
- `DView`. Implements GLUT windows.
- `DLight`. Implements OpenGL lights.

The programming and user manual of DANCE [78] contains a complete discussion of DANCE's architecture and implementation details.

6.5 Who is DANCE for?

In its current state DANCE can be used as a programming and animation environment for research in computer graphics, robotics, biomechanics and the gaming industry. DANCE's modular implementation allows researchers to re-use components that have been developed by the other disciplines. For example, a researcher who is interested in simulating human locomotion, can take advantage of a wide range of existing modules such as humanoid models, simulators, controllers for other motions, filters for geometry input and output, the window mechanism etc. This way not only can they avoid implementing the supporting infrastructure, but they can dress up or enrich their results in a variety of ways. Similarly, DANCE can save time, particularly for graduate students who are interested in physics-based animation and control. According to our own experience it takes approximately 13 months for a master's student to build the necessary infrastructure before he/she can focus on the problem of interest, as tabulated in Table 6.1.

DANCE is a free, portable and open source programming environment that can be used as a classroom tool. Those who teach animation techniques or robotics can use it as the common platform for their classes. The students have the benefit to use a fairly complete tool that comes with a full scale 3D human model and a number of controllers. They can test control or simulation techniques with minimal time spent on issues other than the ones they want to tackle. At the same time, they can share results in their group and also make a contribution to the community by submitting their work to the DANCE repository.

Researchers in biomechanics and human modeling can use it as the common, open source tool that they can either use as common tool to exchange results with their colleagues or tailor it to their specific needs. In addition, Ng-Thow-Hing [77] has developed a powerful muscle model that he might be willing to share upon request with the community.

DANCE has been used so far as follows:

- Virtual puppetry using alternative input devices. University of Toronto, Dept. of Computer Science.
- Interactive physics-based animation of humanoid characters. University of Toronto, Dept. of Computer Science.
- Controller Composition (this Ph.D. thesis). University of Toronto, Dept. of Computer Science.
- Musculotendon modeling, Ng-Thow-Hing's Ph.D. thesis. University of Toronto, Dept. of Computer Science.
- Inverse kinematics techniques. University of Southern California, Dept. of Computer Science.
- Digitized muscle data visualization. University of Toronto, Dept. of Medicine.

We believe that DANCE has the potential to make a significant contribution to the research community either as a standalone tool or as an environment where researchers can build upon the work of others and at the same time contribute their results.

Chapter 7

Results

This chapter discusses the sequences of actions that we are able to achieve using our controller composition method. The SVM-classifier method is used only for the two dimensional robot model for reasons of efficiency. The two dimensional case simulates much faster since it deals with a reduced number of degrees of freedom. The increased simulation efficiency results in a more productive controller design phase. In addition, it significantly improves the time it takes to produce training sets since this is done by repeated forward simulations. Lastly, the support vector machine classification method uses the full state vector of the character and, therefore, it is more efficient for the significantly smaller state vector of the robot model.

The two dimensional case is more robust than the three dimensional one and depends less on the specific character and ground model, as noted in Section 5. However, the sequence we are able to achieve for the full scale three dimensional skeleton model shows that our method can be used successfully for both cases. Inevitably, robust controllers for complex motions will be developed by ourselves or by others. Our system can integrate these controllers as they become available and produce a powerful composite controller.

The next two sections describe in detail our results.

7.1 Robot sequence

The reduced dimensionality of the robot model allowed us to develop a relatively large number of controllers. The sequence shown in Figure 7.1 involves 13 controllers: balance, prone-to-kneel, supine-to-kneel, kneel-to-crouch, crouch-to-stand, stand-to-sit, sit-to-crouch, protective-step, fall, walk, plunge-and-roll, double stance-to-crouch and the default controller. The plunge-and-roll, stand-to-sit, sit-to-crouch and walk controllers bid for control of the character only under the direction of the user. The rest of the controllers act autonomously.

The simulation begins with the robot balancing in place. The user instructs the robot to sit down. The change of the desired goal from "null" to "sit" forces the supervisor controller to invoke the controller selection process. The pre-conditions of the plunge-and-roll, balance and stand-to-sit controllers match the current state of the character. However, the plunge-and-roll controller bids for control only when its goal matches the user specified one. The balance controller bids for control with priority less than 10 since its goal is not "sit". The stand-to-sit controller bids with priority higher than 10 and it therefore becomes the active controller. The robot sits on the toilet and it will stay there under the control of the stand-to-sit controller unless something happens that either changes the state of the character or the desired goal. Soon enough, the user instructs the robot to "lean" which is the goal of the sit-to-crouch controller.

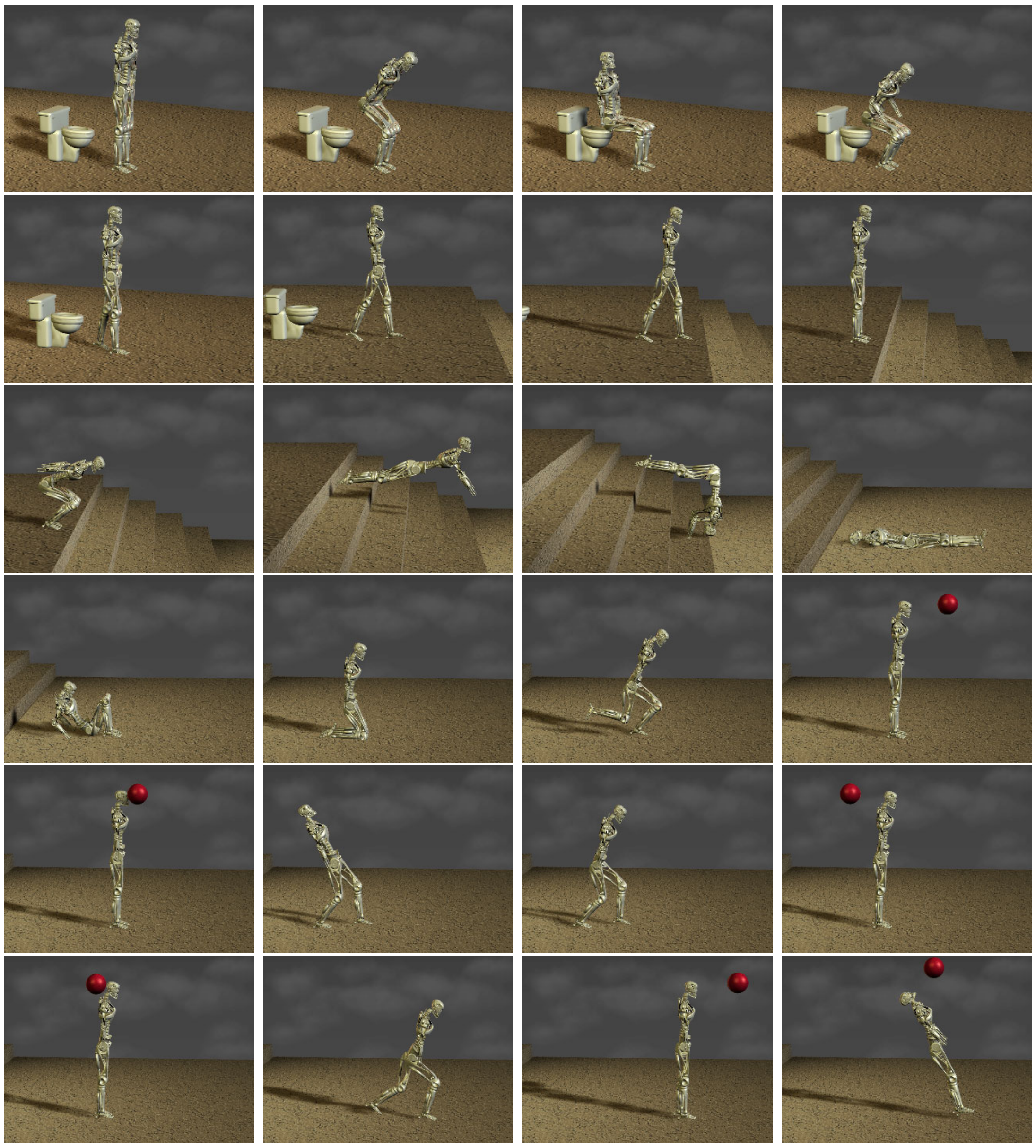


Figure 7.1: The terminator sequence, left to right and top to bottom.

As soon as the robot is in a balanced crouch position the sit-to-crouch controller succeeds and the supervisor controller invokes a controller selection process again. The crouch-to-stand is the only controller that bids for control and after it becomes active it takes the character to a stand position and succeeds. A new selection process allows the balance controller to take over. Then the user instructs the robot to "walk 5" which results in activation of the walk controller. The walk controller completes the five steps bringing the character to a balanced, double stance position at the top of the stairs. The double stance-to-crouch controller takes over and brings the character to a crouch position which allows the crouch-to-stand controller to become active followed by the balance controller. Then the user requests a "dive" which is performed by the plunge-and-roll controller. After the robot rolls over, the default controller takes over until the robot comes to a rest. As soon as the velocity of the center of mass is negligible the supine-to-prone controller becomes active and brings the character to a kneeling position. Then the kneel-to-crouch controller takes over followed by the crouch-to-stand and then the balance controller. The user throws a ball at the head of the robot which throws the robot off balance and results in the activation of the protective-step controller that takes a backward step. The step brings the character to a double stance position which satisfies the pre-conditions of the double stance-to-crouch controller that becomes active and makes the robot assume a crouching position. As before, the crouch-to-stand controller followed by the balancing controller bring the robot to a quiet stance. This time the user throws the ball from the back and at the head of the robot, which results in a forward protective-step behavior, followed by the crouch-to-stand and, finally, the balance controller. The two protective-step behaviors, in addition to a large number of test that we have performed, show that the protective-step, crouch-to-stand and balance controllers are fairly robust. Finally, the user throws the ball with excessive force at the front of the robot's head. The impact induces a large acceleration to the center of mass of the character, which exceeds the pre-conditions of the protective-step controller. The robot reacts immediately with a fall behavior that attempts to use the arms to absorb the impact.

The pre-conditions of the controllers employ the scheme that we described in Section 5.6. If the analytical pre-conditions are satisfied then the SVM-classifier provides the answer. Otherwise the SVM-classifier is never invoked.

This sequence has been created interactively at about 6 frames per second. Unlike the simulation code which is highly optimized, the control code and the ground contact code has not yet been profiled. We expect that profiling of the code will lead to improved efficiency. In addition, employing a semi-implicit method for the integration of the state instead of the current fourth order Runge-Kutta method might further improve the frame rate of the simulation.

7.2 Skeleton sequence

The three dimensional sequence shown in Figure 7.2 is created interactively. The skeleton is equipped with the following controllers: balance, fall, roll-over, prone-crouch, crouch-to-stand and the default controller. All controllers are autonomous in this case; as the skeleton goes through different configurations it automatically reacts to the current situation activating the most appropriate controller among those available. First, the user pushes the character backwards using the *force* actuator described in Section 6.3.3. The composite controller activates a fall behavior that tries to absorb the shock. With the character in a supine configuration, the roll-over controller brings the skeleton to a prone position which makes it possible for the prone-to-crouch controller to take over. When the character reaches a crouching posture, the prone-to-crouch controller succeeds and the crouch-to-stand controller brings the character to

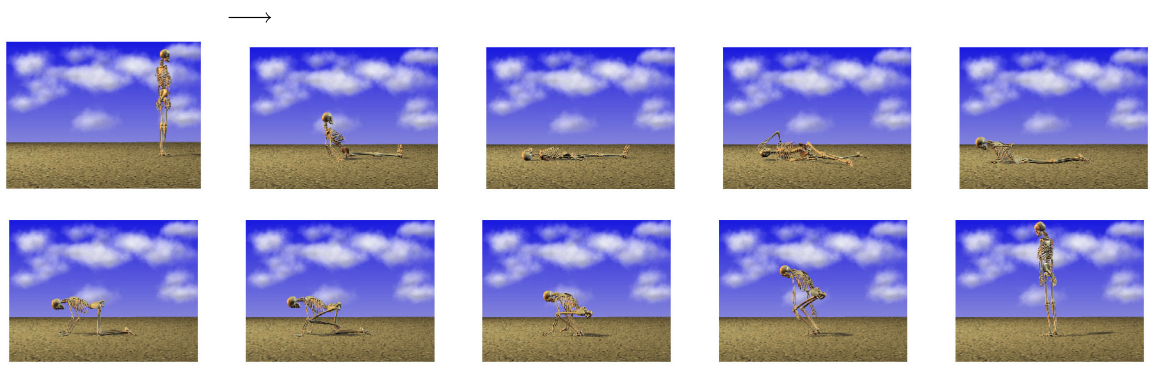

$\longrightarrow$

Figure 7.2: A dynamic “virtual stuntman” falls to the ground, rolls over, and rises to an erect position, balancing in gravity.

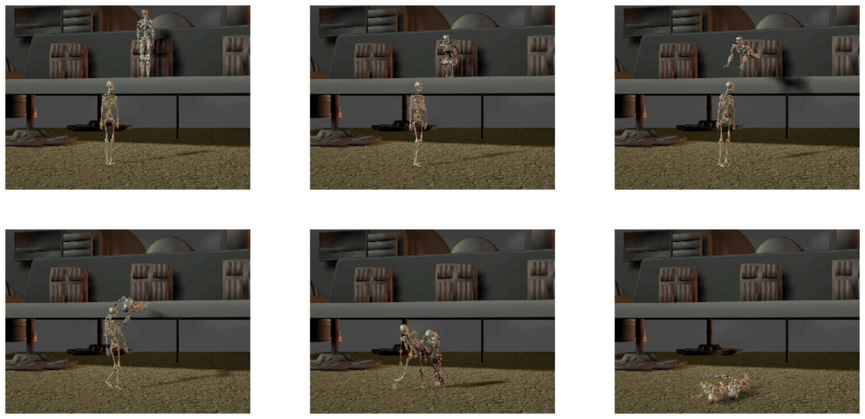


Figure 7.3: Two interacting virtual characters.

an upright position, which allows the balance controller to take over again. This sequence is less robust than the 2D counterpart. The prone-to-crouch and the crouch-to-stand controllers are particularly sensitive to the ground model.

An earlier version of this sequence used a different version of the crouch-to-stand controller which cannot achieve a proper crouching position. The character simply used its left leg to kick up, which resulted in a forward dive that made the character switch to a fall behavior. What was of interest was the ability of the character to fall, roll-over and return to the same kicking position over and over again, as if it was determined to get up, demonstrating that the roll-over controller is fairly robust.

The controllers that take part in this sequence implement manually designed composable APIs based on analytical formulas as described in Chapter 5. The simulation runs at approximately 2 frames per second.

7.3 Multiple characters

Our framework and associated animation system support multiple characters. Each character can have a unique composable controller scheme or share one with other characters. Figure 7.3 shows an example of two interacting three dimensional characters. Currently the motion of

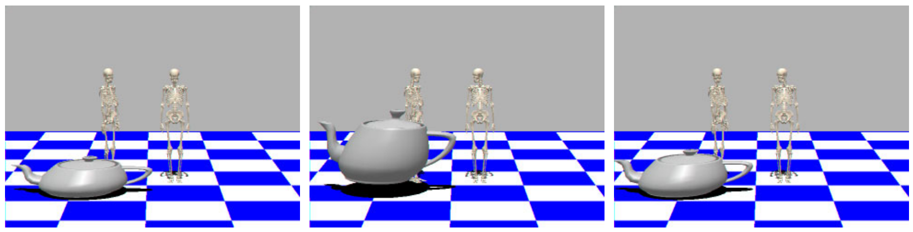


Figure 7.4: Articulated and flexible characters.

each character is computed by a separate simulator. The simulations are performed in a layered fashion. Layered simulation has synchronization problems, Baraff [8], but it works sufficiently well for cases where accuracy is not critical.

The robot standing on the platform, Figure 7.3, is instructed to dive while the skeleton takes a step and balances in place. The timing of the tackle is scripted; the robot is not aware of the presence of the skeleton, it simply collides with the skeleton. The sequence of controllers for the respective characters is as follows:

Skeleton:	Balance → Step → Balance → Protective Step → Fall → Default
Robot:	Balance → Plunge → Default

Given the specialist controllers and the framework to put them together, constructing the tackle example was a simple matter of scripting the scene. The complete script that we used to produce this example is shown in Appendix D for reference.

Our collision actuator considers the complete geometric models of each character. Each of these models has more than sixteen thousand triangles. When the two models collide the simulation becomes very slow, approximately 0.1 frames per second.

Figure 7.4 shows three virtual systems existing in our animation environment. Each system has its own composition framework. The two skeletons are in quiet stance enhanced with Perlin noise. The flexible teapot has been instructed to hop around and has switched from a default controller to a hopping controller. It is part of our future plans to implement a collision actuator that can handle both flexible and rigid bodies.

7.4 Discussion

The sequences that we are able to achieve using our composition framework (Section 3), and the DANCE software system (Section 6) are an indication that the method is a good step towards the development of skillful, autonomous characters. Our approach is bottom up, allowing the incremental development of a wide variety of skills for the simulated characters. The open architecture of DANCE and our composition framework allows practitioners to reuse and build upon existing results and experiment with new techniques on shared models.

At the same time our system supports multiple autonomous systems such as articulated figures, flexible characters and a variety of sophisticated actuators (Section 6), that can co-exist and interact within the same environment, as shown in Figure 7.4.

Chapter 8

Conclusions and Future Work

The challenges of physics-based controller design and the technical obstacles that researchers face when attempting to share their algorithms have hindered progress in the area of physics-based character animation. This thesis has presented a methodology for ameliorating the problem with a framework that facilitates the exchange and composition of controllers. Our framework has been implemented within a freely available system for modeling and animating articulated characters. To our knowledge, our system is the first to demonstrate a dynamic anthropomorphic character with controlled reactions to disturbances or falls in any direction, as well as the ability to pick itself up off the ground in several ways, among other controlled motions. We hope that our system will foster collective efforts among numerous practitioners that will eventually result in complex composite controllers capable of synthesizing a full spectrum of human-like motor behaviors.

Given the enormous challenge of building controllers capable of large repertoires of dynamic human-like motion, it is inevitable that the work presented in this paper is incomplete in many ways. Published control methods for 3D walking, running, and stair climbing make obvious candidates for integration into our system. Coping with variable terrain and dynamic environments are dimensions of added complexity that should provide work for years to come. Automatic parameterization of controllers to variations in character dimensions and mass is a necessary step for having solutions adaptable to a variety of characters. Deriving controllers from motion-capture data is an exciting but difficult prospect, although some progress is already being made in this area. Other methods of "teaching" skills to a dynamic character also warrant investigation.

The work in this thesis can be extended in a variety of ways which are discussed in the next sections.

8.1 Planning

Further advances in the capability of our autonomous characters will require a planning module that is capable of computing the sequence of controllers needed to take the character from a starting state to the desired end state. Currently, this task is left to the user, who is responsible for specifying the sequence of intermediate desired states himself to reach some desired goal state. The planning module must be able to query the available controllers about their pre-conditions and post-conditions and based on the answers compute the appropriate sequence of controllers that will achieve the desired task. In addition, the planning module may have intermediate goals or various constraints that the motion of the character must satisfy. For

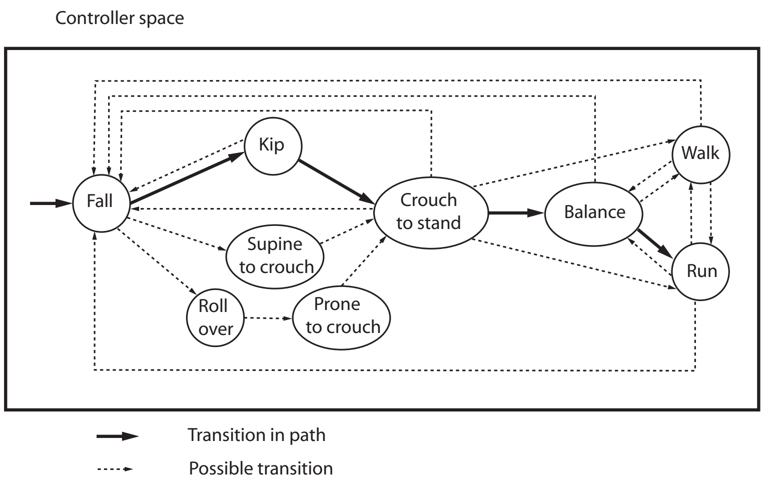


Figure 8.1: A sequence of controllers chosen by a planner.

example, recall the three different ways that our character can transition from a supine position to a standing one: (a) the martial arts kip, (b) the sit-up → crouch → stand-up and (c) supine → prone → crouch → stand-up. A planner that needs to satisfy time constrains, that is have the character standing up as soon as possible, might choose to activate the kip controller. Figure 8.1 shows graphically how a planner might choose a path within the controller state space to achieve a specific result. In this example, we assume that the character has fallen and then needs to get up and go to a specific location as fast as possible. The planner chooses to use the kip motion and then make the character run instead of walk.

Using a planner, as described above, will make the system look much like a tradition finite-state-machine (FSM) controller. However, the pre-conditions and expected performance will still be important in making the system robust in the case of failure. If a failure occurs the motion will have to be replanned.

8.2 Multiple controllers

A serious limitation of the system is that only one controller can be active at each instant. Humans are capable of performing a number of different tasks simultaneously. For example, walking and at the same time waving or looking around. Similarly, most controllers do not need to use all the degrees of freedom of the character. For example, a walking controller does not necessarily need to control the head of the character. In that sense, a walking controller and a controller that implements a visual attention strategy can both be active at the same time. It would therefore be desirable to implement proper resource management, where controllers bid for control of specific groups of degrees of freedom of the character. Resource management in this case is not straightforward, it requires careful design and significant amount of work.

8.3 Training set

The support vector machine classifier described in Chapter 4 separates the states of the character that lead to success from those that lead to failure. In particular, it models the boundary that separates these two classes based on sampled data which is called the training set. As explained in Section 4.5, since we cannot use regular sampling over the entire state space to produce the training set, we resort to stochastic sampling of a few specific regions. It is therefore critical to ensure that these regions and the resulting sample data, include samples from both classes. Our solution is based on the fact that controllers are typically designed to operate around a few nominal states and the only way to guarantee that both classes are represented in the training set is manually.

As future work we would like to implement methods that can sample the state space automatically and guarantee that they include sample points from both classes. One can imaging a process that tunes the boundary placement by applying perturbations on states that cause the controller to succeed. So instead of only perturbing a few user-supplied states, we move from states that cause success along a direction that has better chances of crossing the boundary. We can choose this direction in a variety of ways. A possible candidate is the direction of increasing values of degrees of freedom associated with "important" joints. We consider important joints those that are associated with large body parts, such as the legs and trunk. Thus, we can start the sampling process at a nominal successful state, and then constantly increase the angle of one important joint while stochastically perturbing the rest until we get a failure. We record the state that produced the failure and we stochastically sample the region around that state. Then we can return to the nominal state and repeat the same process using another important joint angle until each has been explored in turn.

The notion of important degrees of freedom leads to another issue of interest. How can we identify automatically which degrees of freedom affect a specific motion? For example, the motion of the hands has much less effect during quiet stance compared to the motion of the whole arm. In contrast, the hands are very important when the character moves from a prone to a crouch position. If we know in advance the degrees of freedom that are important for a motion, then we can concentrate our sampling efforts on those and possibly reduce the state space of the character.

8.4 Expected performance and pre-conditions.

In our current implementations, controllers have only one entry point. Pre-conditions are defined only at the point that the controllers consider their starting point. In general, controllers take the character through a series of configurations within its state space. It would be useful if the controllers could start operating with any one of these configurations as the starting point. In that case, the pre-conditions and the expected performance would be the same, in the sense that the expected performance would be a series of pre-conditions.

It is worth noting that the unique entry point is a limitation of the current controller design and not of the composition framework. As far as the latter is concerned, it is up to the controllers themselves to decide when they can bid for control of the character.

8.5 Additional testing

It would be interesting to perform exhaustive tests on our controllers and their computed pre-conditions. To this extent, it might be useful to develop a screensaver version of our system that uses the spare machine cycles to perform tests over long periods of time. In this case, the system can be driven by random inputs from a fake user and we can see how often the system would fail.

8.6 Future: Intelligent agents

The work in this thesis is a first step towards developing skillful autonomous characters. It will take considerable amounts of work and collaboration between researchers in graphics, biomechanics, and robotics before we can implement a physics-based simulated character that can play a dynamic sport such as tennis. At the same time, advances in robotics, graphics and artificial intelligence have clearly shown that developing virtual agents with complex motor skills and interesting levels of intelligence is not as elusive as it once was. The future of intelligent agents looks promising.

Appendix A

SVM^{light} parameters

SVM^{light} [57] is an excellent implementation of the support vector machine classification method. Below are the parameters that the user can adjust to control the performance of the produced classifier. We present them exactly as they appear when the program is invoked without any input.

```
SVM-light V3.02: Support Vector Machine, learning module     16.11.99

Copyright: Thorsten Joachims, thorsten@ls8.cs.uni-dortmund.de

This software is available for non-commercial use only. It must not
be modified and distributed without prior permission of the author.
The author is not responsible for implications from the use of this
software.

   usage: svm_learn [options] example_file model_file

Arguments:
         example_file-> file with training data
         model_file  -> file to store learned decision rule in
General options:
         -?          -> this help
         -v [0..3]   -> verbosity level (default 1)
Learning options:
         -c float    -> C: trade-off between training error
                        and margin (default 1000)
         -j float    -> Cost: cost-factor, by which training errors on
                        positive examples outweight errors on negative
                        examples (default 1)
         -b [0,1]    -> use biased hyperplane (i.e. x*w+b>0) instead
                        of unbiased hyperplane (i.e. x*w>0) (default 1)
         -i [0,1]    -> remove inconsistent training examples
                        and retrain (default 0)
Transduction options:
         -p [0..1]   -> fraction of unlabeled examples to be classified
                        into the positive class (default is the ratio of
                        positive and negative examples in the training data)
Kernel options:
         -t int      -> type of kernel function:
                        0: linear (default)
                        1: polynomial (s a*b+c)^d
                        2: radial basis function exp(-gamma ||a-b||^2)
                        3: sigmoid tanh(s a*b + c)
                        4: user defined kernel from kernel.h
         -d int      -> parameter d in polynomial kernel
         -g float    -> parameter gamma in rbf kernel
         -s float    -> parameter s in sigmoid/poly kernel
         -r float    -> parameter c in sigmoid/poly kernel
         -u string   -> parameter of user defined kernel
Optimization options:
         -q [2..400] -> maximum size of QP-subproblems (default 10)
         -m [5..]    -> size of cache for kernel evaluations in MB (default 40)
                        The larger the faster...
         -e float    -> eps: Allow that error for termination criterion
                        [y [w*x+b] - 1] >= eps (default 0.001)
         -h [5..]    -> number of iterations a variable needs to be
                        optimal before considered for shrinking (default 100)
         -f [0,1]    -> do final optimality check for variables removed
                        by shrinking. Although this test is usually
                        positive, there is no guarantee that the optimum
                        was found if the test is omitted. (default 1)
Output options:
         -l char     -> file to write predicted labels of unlabeled
                        examples into after transductive learning
         -a char     -> write all alphas to this file after learning
                        (in the same order as in the training set)
```

Appendix B

SD/Fast description file for our 3D character

```
# This is the system description file for object skel18

gravity = 0 0 0
language = c

double
prefix = skel18

body = Hip inb = $ground joint = free
        mass = 16.610000
        inertia = 0.180000 0.160000 0.230000
        inbtojoint = 0.000000 1.131000 0.005200
        bodytojoint = 0.000000 0.120600 0.050000
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 1.000000 0.000000
        pin = 0.000000 0.000000 1.000000
body = Trunk_comp inb = Hip joint = gimbal
        mass = 29.270000
        inertia = 0.630000 0.320000 0.730000
        inbtojoint = 0.000000 0.120600 0.050000
        bodytojoint = 0.000000 -0.146250 0.035100
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
body = Neck inb = Trunk_comp joint = gimbal
        mass = 1.000000
        inertia = 0.006000 0.001000 0.006000
        inbtojoint = 0.000000 0.246350 -0.001300
        bodytojoint = 0.000000 -0.044200 -0.006500
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
```

```
body = Head_comp inb = Neck joint = pin
        mass = 5.890000
        inertia = 0.033000 0.023000 0.030000
        inbtojoint = 0.000000 0.044200 0.006500
        bodytojoint = 0.000000 -0.078000 -0.031200
        pin = 1.000000 0.000000 0.000000
body = Left_Shoulder inb = Trunk_comp joint = ujoint
        mass = 2.790000
        inertia = 0.005000 0.025000 0.025000
        inbtojoint = 0.176800 0.189150 0.001300
        bodytojoint = -0.132600 0.000000 -0.001300
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
body = Left_Forearm inb = Left_Shoulder joint = ujoint
        mass = 1.210000
        inertia = 0.001200 0.005400 0.005000
        inbtojoint = 0.132600 0.000000 0.001300
        bodytojoint = -0.137800 -0.003900 0.000000
        pin = 0.000000 1.000000 0.000000
        pin = 1.000000 0.000000 0.000000
body = Left_Hand inb = Left_Forearm joint = ujoint
        mass = 0.550000
        inertia = 0.000500 0.002000 0.001600
        inbtojoint = 0.137800 0.003900 0.000000
        bodytojoint = -0.065000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
body = Right_Shoulder inb = Trunk_comp joint = ujoint
        mass = 2.790000
        inertia = 0.005000 0.025000 0.025000
        inbtojoint = -0.176800 0.189150 0.001300
        bodytojoint = 0.132600 0.000000 -0.001300
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
body = Right_Forearm inb = Right_Shoulder joint = ujoint
        mass = 1.210000
        inertia = 0.001200 0.005400 0.005000
        inbtojoint = -0.132600 0.000000 0.001300
        bodytojoint = 0.137800 -0.003900 0.000000
        pin = 0.000000 1.000000 0.000000
        pin = 1.000000 0.000000 0.000000
body = Right_Hand inb = Right_Forearm joint = ujoint
        mass = 0.550000
        inertia = 0.000500 0.002000 0.001600
        inbtojoint = -0.137800 0.003900 0.000000
        bodytojoint = 0.065000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
```

```
body = Left_Thigh inb = Hip joint = gimbal
        mass = 8.350000
        inertia = 0.160000 0.025000 0.150000
        inbtojoint = 0.111800 -0.040600 0.005800
        bodytojoint = 0.016900 0.237900 -0.014300
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
body = Left_Shin inb = Left_Thigh joint = pin
        mass = 4.160000
        inertia = 0.056000 0.007000 0.055000
        inbtojoint = -0.016900 -0.237900 0.014300
        bodytojoint = 0.007800 0.215800 0.003900
        pin = 1.000000 0.000000 0.000000
body = Left_Foot inb = Left_Shin joint = ujoint
        mass = 1.340000
        inertia = 0.007500 0.007000 0.001800
        inbtojoint = -0.007800 -0.215800 -0.003900
        bodytojoint = 0.000000 0.044200 -0.039000
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
body = Right_Thigh inb = Hip joint = gimbal
        mass = 8.350000
        inertia = 0.160000 0.025000 0.150000
        inbtojoint = -0.111800 -0.040600 0.005800
        bodytojoint = -0.016900 0.237900 -0.014300
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
        pin = 0.000000 1.000000 0.000000
body = Right_Shin inb = Right_Thigh joint = pin
        mass = 4.160000
        inertia = 0.056000 0.007000 0.055000
        inbtojoint = 0.016900 -0.237900 0.014300
        bodytojoint = -0.007800 0.215800 0.003900
        pin = 1.000000 0.000000 0.000000
body = Right_Foot inb  = Right_Shin joint = ujoint
        mass = 1.340000
        inertia = 0.007500 0.007000 0.001800
        inbtojoint = 0.007800 -0.215800 -0.003900
        bodytojoint = 0.000000 0.044200 -0.039000
        pin = 1.000000 0.000000 0.000000
        pin = 0.000000 0.000000 1.000000
```

Appendix C

SD/Fast description file for our 2D character

```
# This is the system description file for object human2D

gravity = 0 0 0
language = c

double
prefix = human2D

body = Hip inb = $ground joint = planar
        mass = 16.610000
        inertia = 0.180000 0.00000 0.00000
        inbtojoint = 0.000000 1.131000 0.005200
        bodytojoint = 0.000000 0.120600 0.050000
        pin = 0.000000 1.000000 0.00000
        pin = 0.000000 0.000000 1.00000
        pin = 1.000000 0.000000 0.00000
body = Trunk_comp inb = Hip joint = pin
        mass = 29.270000
        inertia = 0.630000 0.000000 0.000000
        inbtojoint = 0.000000 0.120600 0.050000
        bodytojoint = 0.000000 -0.146250 0.035100
        pin = 1.000000 0.000000 0.000000
body = Neck inb = Trunk_comp joint = pin
        mass = 1.000000
        inertia = 0.006000 0.000000 0.000000
        inbtojoint = 0.000000 0.246350 -0.001300
        bodytojoint = 0.000000 -0.044200 -0.006500
        pin = 1.000000 0.000000 0.000000
body = Head_comp inb = Neck joint = pin
        mass = 5.890000
        inertia = 0.033000 0.000000 0.000000
        inbtojoint = 0.000000 0.044200 0.006500
        bodytojoint = 0.000000 -0.078000 -0.031200
        pin = 1.000000 0.000000 0.000000
body = Left_Shoulder inb = Trunk_comp joint = pin
        mass = 2.790000
        inertia = 0.025000 0.000000 0.000000
        inbtojoint = 0.176800 0.189150 0.001300
        bodytojoint = 0.000000 0.132600 -0.001300
        pin = 1.000000 0.000000 0.000000
body = Left_Forearm inb = Left_Shoulder joint = pin
        mass = 1.210000
        inertia = 0.00540 0.000000 0.000000
        inbtojoint = 0.000000 -0.132600 0.001300
        bodytojoint = 0.003900 0.137800  0.000000
        pin = 1.000000 0.000000 0.000000
body = Right_Shoulder inb = Trunk_comp joint = pin
        mass = 2.790000
        inertia = 0.025000 0.000000 0.000000
        inbtojoint = -0.176800 0.189150 0.001300
        bodytojoint = 0.000000 0.132600 -0.001300
        pin = 1.000000 0.000000 0.000000
body = Right_Forearm inb = Right_Shoulder joint = pin
        mass = 1.210000
        inertia = 0.005400 0.000000 0.000000
        inbtojoint = 0.000000 -0.132600 0.001300
        bodytojoint = -0.003900 0.137800 0.000000
        pin = 1.000000 0.000000 0.000000
body = Left_Thigh inb = Hip joint = pin
        mass = 8.350000
        inertia = 0.160000 0.000000 0.000000
        inbtojoint = 0.111800 -0.040600 0.005800
        bodytojoint = 0.016900 0.237900 -0.014300
        pin = 1.000000 0.000000 0.000000
body = Left_Shin inb = Left_Thigh joint = pin
        mass = 4.160000
        inertia = 0.056000 0.000000 0.000000
        inbtojoint = -0.016900 -0.237900 0.014300
        bodytojoint = 0.007800 0.215800 0.003900
        pin = 1.000000 0.000000 0.000000
body = Left_Foot inb = Left_Shin joint = pin
        mass = 1.340000
        inertia = 0.007500 0.000000 0.000000
        inbtojoint = -0.007800 -0.215800 -0.003900
        bodytojoint = 0.000000 0.044200 -0.039000
        pin = 1.000000 0.000000 0.000000
body = Right_Thigh inb = Hip joint = pin
        mass = 8.350000
        inertia = 0.160000 0.000000 0.000000
        inbtojoint = -0.111800 -0.040600 0.005800
        bodytojoint = -0.016900 0.237900 -0.014300
        pin = 1.000000 0.000000 0.000000
body = Right_Shin inb = Right_Thigh joint = pin
        mass = 4.160000
        inertia = 0.056000 0.000000 0.000000
        inbtojoint = 0.016900 -0.237900 0.014300
        bodytojoint = -0.007800 0.215800 0.003900
        pin = 1.000000 0.000000 0.000000
body = Right_Foot inb = Right_Shin joint = pin
        mass = 1.340000
        inertia = 0.007500 0.000000 0.000000
        inbtojoint = 0.007800 -0.215800 -0.003900
        bodytojoint = 0.000000 0.044200 -0.039000
        pin = 1.000000 0.000000 0.000000
```

Appendix D

DANCE script for the tackle example

This is the DANCE tcl script that was used to produce the tackle example shown in Figure 7.3. Because collisions are expensive we turned them one right before the two characters collide.

```
set wdir $env(DANCE_DIR)/run/skel18

#-------------------------------------
# load the systems and the environment
#-------------------------------------
# load character 1
source $wdir/setUpSkel.tcl
# load character 2
source $wdir/setUpSkel2.tcl

# load the simulators
instance SdfastSimul Skel18Simul1 skel18
simulator Skel18Simul1 f dt 0.00001
instance SdfastSimul Skel18Simul2 skel18_2
simulator Skel18Simul2 f dt 0.00001

# set the view
source $wdir/scene.tcl

# set the ground
instance PlaneGround ground
actuator ground apply all
actuator ground set model 27424 805 400

# set the platform for the robot to stand on
instance PlaneGround platform
actuator platform set geometry 1.0 -0.1 -2.0 1.0 -0.1 -0.44 2.0  \
-0.1 -0.44 2.0 -0.1 -2.0 10 10
actuator plarform apply skel18
actuator platform set model 27424 805 400
```

```
# set gravity
instance FieldActuator grav
actuator grav set field 0 -9.8 0
actuator grav apply all

#--------------------------------
# load the supervisor controllers
#--------------------------------
# CAUTION: First load super then a generic posecontroller
#        then  the rest and add the default controller last
instance SupervisorController super1
actuator super1 apply skel18
instance SupervisorController super2
actuator super2 apply skel18_2

# load the PoseController2 plugin
plugin PoseController2

# load the specialist controllers
instance FallController2 fallCon1
instance FallController2 fallCon2
instance ProtectStepController protectStep2
instance StepController step2
instance DefaultController defaultCon1
instance DefaultController defaultCon2
instance InvPendController invpend1
instance InvPendController invpend2
instance DiveOnHumanController dive1
actuator dive1 lean_threshold 0.8

#set parameters for the balance controllers
actuator invpend1  param 70 10 1.0 0.06 0.06
actuator invpend2  param 70 10 1.0 0.06 0.06

# put the specialist controllers under the control
# of the appropriate supervisor controller
actuator super1 add invpend1
actuator super1 add dive1
actuator super1 add fallCon1
actuator super1 add defaultCon1

actuator super2 add invpend2
actuator super2 add step2
actuator super2 add protectStep2
actuator super2 add fallCon2
actuator super2 add defaultCon2
```

```
# ------------------------------------
# set up collision for all geometries
#-------------------------------------
# that is compute the RAPID structures
set GeomList [show geometry]
foreach geomName $GeomList {
    geometry $geomName setupCollision
}

# set up the parameters of collision actuator
instance CollisionActuator collision
actuator collision apply skel18
actuator collision apply skel18_2
actuator collision stiffness 1000.0
actuator collision damping 100.0
actuator collision friction 1.0
# turn collisions off for now
actuator collision other_collision off
actuator collision self_collision off

# -------------------
# set up the scenario
# -------------------

# indicate to the super2 controller that want the skeleton to take
# a step
actuator super2 desired step

# set the following events to take place later
# set the desired goal for the robot to "dive" at 0.05 secs after
# the simulation starts
system skel18 simul_event add 0.05 "actuator super1 desired dive_on_human"
# turn collisions on at 2.8 seconds after the simulation starts,
# which is before the two characters collide
system skel18 simul_event add 2.80 "actuator collision other_collision on"
system skel18 simul_event add 2.80 "actuator collision self_collision on"

# -----------------------------------------
# Everything is set so start the simulation
# -----------------------------------------
simul start
```

Bibliography

[1] Alias/Wavefront, a division of Silicon Graphics Canada Limited, Toronto, Ontario. *Maya*, 1997.

[2] Ronald C. Arkin. *Behavioral Robotics.* MIT Press, 1998.

[3] W. W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. *Proceedings of Graphics Interface '85*, pages 407–415, 1985.

[4] N. Badler, C. Phillips, and B. Webber. *Simulating Humans: Computer Graphics, Animation, and Control.* Oxford University Press, 1993.

[5] N. I. Badler, B. Barsky, and D. Zeltzer. *Making Them Move.* Morgan Kaufmann Publishers Inc., 1991.

[6] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating rigid bodies. *Proceedings of ACM SIGGRAPH: Computer Graphics*, 23(3):223–232, July 1989.

[7] David Baraff. Linear-time dynamics using lagrange multipliers. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, pages 137–146, 1996.

[8] David Baraff and Andrew Witkin. Partitioned dynamics. Technical report, Carnegie Mellon University, 1997.

[9] David Baraff and Andrew Witkin. Large steps in cloth simulation. *Computer Graphics*, 32(Annual Conference Series):43–54, August 1998.

[10] Bell-Labs. SVT applet. `http://svm.research.bell-labs.com/SVT/SVMsvt.html`.

[11] B.W. Bergemann and H.C. Sorenson. Dynamic analysis of the kip on the high bar. In J. Terauds and D.B. (Hrsg.) Daniels, editors, *Science in Gymnastics*, pages 44–54. Academic Publishers, Del Mar, California, 1979.

[12] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of robotics and automation*, RA–2(1):14–23, 1986.

[13] A. Bruderlin and L. Williams. Motion signal processing. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 97–104, August 1995.

[14] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):955–974, 1998.

[15] R. R Burridge, A. A. Rizzi, and D. E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18(6):534–555, June 1999.

[16] Tolga Capin, Igor Pandzic, Nadia Magnenat Thalmann, and Daniel Thalmann. *Avatars in Networked Virtual Environments*. John Wiley & Sons, 1999.

[17] Eva Cerezo, Alfredo Pina, and Fransisco J. Serón. Motion and behaviour modelling: state of the art and new trends. *The Visual Computer*, 15:124–146, 1999.

[18] Stephen Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 219–228. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[19] Chee-Meng Chew and Gill A. Pratt. A general control architecture for dynamic bipedal walking. In *Proceedings of the 2000 IEEE International Conference and Automation*, pages 3990–3996, San Francisco, CA, April 2000.

[20] Nello Christianini and John Shawe-Taylor. *Support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[21] M. F. Cohen. Interactive spacetime control for animation. *Proceedings of ACM SIGGRAPH: Computer Graphics*, 26(2):293–302, July 1992.

[22] P. Corke. A robotics toolbox for matlab. *IEEE Robot. Automat. Mag.*, 3(1):24–32, 1996.

[23] B. L. Day, M. J. Steiger, P. D Thompson, and C. D. Marsden. Effect of vision and stand width on human body motion when standing: implications for afferent control of lateral sway. *Journal of Physiology*, 469:479–499, 1993.

[24] M. C. Do, Y. Breniere, and P. Brenguier. A biomechanical study of balance recovery during the fall forward. *Journal of Biomechanics*, 15(12):933–939, 1982.

[25] R C. Dorf. *Modern Control Systems*. Addison-Wesley,Reading, Massachusetts, 1989.

[26] R. O. Duda and P. E Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[27] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, 1997.

[28] François Faure. Fast iterative refinement of articulated solid dynamics. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):268–276, July/September 1999.

[29] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, 1987.

[30] R. C Fitzpatrick, R. L. Gorman, and D. Burke. Postural proprioceptive reflexes in standing human subjects: bandwidth of response and transmission characteristics. *Journal of Physiology*, 458:69–83, 1992.

[31] R. C Fitzpatrick and D. I. McCloskey. Proprioceptive, visual and vestibular thresholds for the perception of sway during standing in humans. *Journal of Physiology*, 478(1):173–186, 1994.

[32] R. C Fitzpatrick, J. L. Taylor, and D. I. McCloskey. Ankle stiffness of standing humans in response to imperceptible perturbation: reflex and task-dependent components. *Journal of Physiology*, 454:533–547, 1992.

[33] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proceedings of SIGGRAPH 1999*, pages 29–38, Los Angeles, CA, August 1999.

[34] Plamen Gatev, Sherry Thomas, Thomas Kepple, and Mark Hallet. Feedforward ankle strategy of balance during quiet stance in adults. *Journal of Physiology*, 514(3):915–928, 1999.

[35] M. Gienger, K. Löffler, and F. Pfeiffer. A biped robot that jogs. In *Proceedings of the 2000 IEEE International Conference and Automation*, pages 3334–3339, San Francisco, CA, April 2000.

[36] Michael Gleicher. Retargeting motion to new characters. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 33–42. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[37] H. Goldstein. *Classical Mechanics.* Addison-Wesley, 2nd edition, 1980.

[38] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 1996*, pages 171–180, 1996.

[39] Larry Gritz and James K. Hahn. Genetic programming evolution of controllers for 3-D character animation. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 139–146, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[40] UNC Collide Research Group. Collision detection software packages. `www.cs.unc.edu/~geom/collide/packages.shtml`, 2000.

[41] R. Grzeszczuk and D. Terzopoulos. Automated learning of muscle-based locomotion through control abstraction. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 63–70, August 1995.

[42] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 9–20, July 1998.

[43] Paul S. HeckBert, editor. *Graphics Gems IV.* Academic Press, 1994.

[44] J. K. Hodgins. Biped gait transitions. *Proceedings of 1991 IEEE International Conference on Robotics and Automation*, pages 2092–2097, April 1991.

[45] J. K. Hodgins and M. H. Raibert. Biped gymnastics. *International Journal of Robotics Research*, 9(2):115–132, April 1990.

[46] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. *Proceedings of SIGGRAPH 95, ACM Computer Graphics*, pages 71–78, 1995.

[47] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 71–78, August 1995.

[48] Jessica Hodgins and Zoran Popovic. Animating humans by combining simulation and motion capture. Course notes CDROM #2. ACM SIGGRAPH 2000, `www.siggraph.org`.

[49] Michael G. Hollars, Dan E. Rosenthal, and Michael A. Sherman. Sd/fast. Symbolic Dynamics, Inc., 1991.

[50] E. T Hsiao and S. N Robinovitch. Common protective movements govern unexpected falls from standing height. *Journal of biomechanics*, 31:1–9, 1998.

[51] Q. Huang, K. Kaneko, K. Yokoi, S. Kajita, T. Kotoku, N. Koyachi, H. Arai, N. Imamura, K. Komoriya, and K. Tanie. Balance control of a biped robot combining off–line pattern with real–time modification. In *Proceedings of the 2000 IEEE International Conference and Automation*, pages 3346–3352, San Francisco, CA, April 2000.

[52] Animats Inc. Falling bodies. `www.animats.com`.

[53] Boston Dynamics Inc. The digital biomechanics laboratory. `www.bdi.com`, 1998.

[54] Havok Inc. Havok sdk. `www.havok.com`.

[55] Sarcos Inc. `www.sarcos.com`.

[56] Kenji Inoue, Haruyuki Yoshida, Tatsuo Arai, and Yasushi Mae. Mobile manipulation of humanoids-real–time control based on manipulability and stability. In *Proceedings of the 2000 IEEE International Conference and Automation*, pages 2217–2222, San Francisco, CA, April 2000.

[57] T. Joachims. Making large-scale svm learning practical. advances in kernel methods. In B. Schölhopf, C. Burges, and A. Smola, editors, *Support Vector Learning*. MIT-Press, 1999. http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_99a.pdf.

[58] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[59] Taku Komura, Yoshihisa Shinagawa, and Tosiyasu L. Kunii. Calculation and visualization of the dynamic ability of the human body. *The journal of visualization and computer animation*, 10:57–78, 1999.

[60] MIT Leg lab. `www.ai.mit.edu/projects/leglab/robots/robots.html`.

[61] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive control for physically-based animation. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 201–208. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[62] Joseph F. Laszlo, Michiel van de Panne, and Eugene Fiume. Limit cycle control and its application to the animation of balancing and walking. *Proceedings of SIGGRAPH 96*, pages 155–162, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[63] Z. Liu, S. J. Gortler, and M. F. Cohen. Hierarchical spacetime control. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 293–302, August 1994.

[64] R. Bowen Loftin, James, C. Maida, and Jian Yang. Inverse kinematics of the human arm. Annual report, The Institute for Space Systems Operations, 1996-1997. `www.isso.uh.edu/publications/A9697/9697-8.html`.

[65] Honda Motor Co. Ltd. `www.honda.co.jp/english/technology/robot/`.

[66] Sony Co. Ltd. `www.sony.co.jp/en/SonyInfo/News/Press/200011/00-057E2/`.

[67] David K. MacAllister. Particle system api. `www.cs.unc.edu/~davemc/Particle`, 1997.

[68] Pattie Maes and Rondey A. Brooks. Learning to coordinate behaviors. In *Proceedings Eight National Conference on Arifucual Intelligence, AAAI 1990*, volume 2, pages 796–802, Boston, July 1990. MIT Press.

[69] J. B. Marion and S. T. Thornton. *Classical Dynamics of Particles and Systems*. Harcourt Brace Jovanovich, Publishers, third edition, 1988.

[70] Mathengine simulation software. Mathengine PLC. Oxford, UK.

[71] T. A. McMahon. The role of compliance in mammalian running gaits. *Journal of Experimental Biology*, 115:263–282, 1985.

[72] G. S. P. Miller. The motion dynamics of snakes and worms. *Proceedings of ACM SIGGRAPH: Computer Graphics*, 22(4):169–178, August 1988.

[73] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(2), 1996.

[74] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California at Berkeley, 1996.

[75] Brian Mirtich. V-Clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[76] J. Nethery. *Robotica: A structured environment for computer-aided design and anlysis of robots*. PhD thesis, University of Illinois, 1993.

[77] Victor Ng-Thow-Hing. *Anatomically-based models for physical and geometric resconstructions of animals*. PhD thesis, University of Toronto, Toronto,Canada, 2000.

[78] Victor Ng-Thow-Hing and Petros Faloutsos. Dance user and programming manual. `www.dgp.toronto.edu/dgp/software/dance/danceDownload.html`, 2000.

[79] J. T. Ngo and J. Marks. Spacetime constraints revisited. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 343–350, August 1993.

[80] Yi-Chung Pai and Kamran Iqbal. Simulated movement termination for balance recovery: can movement strategies be sought to maintain stability in the presence of slipping or forced sliding? *Journal of biomechanics*, 32:779–786, 1999.

[81] Yi-Chung Pai and James Patton. Center of mass velocity–position predictions for balance control. *Journal of biomechanics*, 30(4):347–354, 1997.

[82] Marcus G. Pandy and Frank C. Anderson. Three-dimensional computer simulation of jumping and walking using the same model. In *Proceedings of the VIIth International Symposium on Computer Simulation in Biomechanics*, August 1999.

[83] Marcus G. Pandy and Felix E. Zajac. Optimal muscular coordination strategies for jumping. *Journal of Biomechanics*, 24(1):1–10, 1991.

[84] Marcus G. Pandy, Felix E. Zajac, Eunsup Sim, and William S. Levine. An optimal control model for maximum-height human jumping. *Journal of Biomechanics*, 23(12):1185–1198, 1990.

[85] Elisabetta Papa and Aurelio Cappozzo. Sit–to–stand motor strategies investigated in able-bodied your and elderly subjects. *Journal of biomechanics*, 33:1113–1122, 2000.

[86] K. Perlin. An image synthesizer. *Computer Graphics*, 19(3):287–296, July 1985.

[87] Jovan Popovic, Steven M. Seitz, Michael Erdmann, Zoran Popovic, and Andrew Witkin. Interactive manipulation of rigid body simulations. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 209–218. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[88] M. H. Raibert. *Legged Robots that Balance*. MIT Press, 1986.

[89] W. T. Reeves. Particle systems — A technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

[90] J. Reich, S. Swoboda, R. Steiner, and W. J. Daunicht. Biomex 2 – an environment for complex biomechanical simulations. `isb.ri.ccf.org/tgcs/iscsb7/abstracts/reich.pdf`.

[91] J. A. Reicler and F. delcomyn. Dynamics simulation and controller interfacing for legged robots. *The International Journal of Robotics Research*, 19(1):41–57, 2000.

[92] RoboCup. `www.robocup.org`.

[93] R. Romick-Allen and A. B. Schultz. Biomechanics of reactions to impeding falls. *Journal of Biomechanics*, 21(7):591–600, 1988.

[94] Richard A. Schmidt and Craig A. Wrisberg. *Motor Learning and Performance*. Human Kinetics, Champaign, 2nd edition, 2000.

[95] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH 86 Proceedings)*, volume 20, pages 151–160, August 1986.

[96] K. Sims. Evolving virtual creatures. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 15–22, 1994.

[97] Cecile Smeesters, Wilson C. Hayes, and Thomas A. McMahon. Determining fall direction and impact location for various disturbances and gait speeds using the articulated total body model. In *Proceedings of the VIIth International Symposium on Computer Simulation in Biomechanics*, August 1999.

[98] T. Spägele, A. Kristner, and A. Gollhofer. Modelling, simulation and optimization of the human vertical jump. *Journal of biomechanics*, 32:521–530, 1999.

[99] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In Andrew Glassner, editor, *Computer Graphics (SIGGRAPH 94 Proceedings)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50, July 1994.

[100] M. van de Panne and E. Fiume. Sensor-actuator networks. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 335–342, August 1993.

[101] M. van de Panne, R. Kim, and E. Fiume. Synthesizing parameterized motions. *Fifth Eurographics Workshop on Animation and Simulation, at Oslo*, September 1994.

[102] M. van de Panne, R. Kim, and E. Fiume. Virtual wind-up toys for animation. *Graphics Interface*, pages 208–315, 1994.

[103] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable motion synthesis using state-space controllers. *Computer Graphics (SIGGRAPH 90 Proceedings)*, 24(4):225–234, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.

[104] Michiel van de Panne, Eugune Fiume, and Zvonko G. Vranesic. A controller for the dynanic walk of a biped across variable terrain. In *Proceedings of the 31rst Conference on decision and control*, pages 2668–2673, Tuscon, Arizona, 1992.

[105] A. J. van den Kroonenberg, W. C. Hayes, and T. A. McMahon. Hip impact velocities and body configurations for voluntary falls from standing height. *Journal of Biomechanics*, 29(6):807–811, 1996.

[106] Herman van der Kooij, Bart Koopman, Ron Jacobs, Thomas Mergner, and Henk Grootenboer. Quantification of sesonry information in human balance control. In *Proceedings of the 20th Annual International COnference of the IEEE Engineering in Medicine and Biology Society*, volume 20, pages 2393–2396, 1998.

[107] V. Vapnik. *Estimation of Dependecies Based on Empirical Data (in Russian).* Nauka, Moscow, 1979. English translation Springer Verlag, New York, 1982.

[108] Jane Wilhelms and Brian A. Barsky. Using dynamic analysis to animate articulated bodies such as humans and robots. In *Graphics Interface '85*, pages 97–104, May 1985.

[109] David A. Winter. Anthropometry. In *Biomechanics and Motor Control of Human Movement*, chapter 3, pages 51–74. John Wiley and Sons, Inc., second edition, 1990.

[110] A. Witkin and M. Kass. Spacetime constraints. *Proceedings of ACM SIGGRAPH: Computer Graphics*, 22(4):159–168, August 1988.

[111] A. Witkin and Z. Popovic. Motion warping. *Proceedings of ACM SIGGRAPH: Computer Graphics*, pages 105–108, August 1995.

[112] W. L. Wooten and J. K. Hodgins. Animation of human diving. *Computer Graphics Forum*, 15(1):3–13, March 1996.

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

[114] Ge Wu. Distinguishing fall activities from normal activities by velocity characteristics. *Journal of Biomechanics*, 33:1497–1500, 2000.

[115] D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, pages 53–59, November 1992.